НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інофрматики та обчислювальної техніки

Кафедра обчислювальної техніки

«На правах рукопису»
УДК __*004.02*____

До захисту допущено:

Завідувач кафедри

_____ Сергій Стіренко

«___»_____20__ р.

Магістерська дисертація

на здобуття ступеня магістра

за освітньо-професійною програмою «Інженерія програмного забезпечення комп'ютерних систем»

зі спеціальності 121 «Інженерія програмного забезпечення»

на тему: «Метод резервування та відновлення даних при їх зберіганні на віддалених носіях»

Виконав (-ла):
студент (-ка) VI курсу, групи ІП-94мп
Ковбаса Святослав Ігорович _____

Керівник:
Доцент, кандидат технічних наук
Марковський Олександр Петрович _____

Консультант з нормоконтролю:
Професор, доктор технічних наук,
Жабін Валерій Іванович _____

Рецензент:
Декан ФПМ, доктор технічних наук, професор
Дичка Іван Андрійович _____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____

Київ – 2020 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет (інститут) *Інформатики та обчислювальної техніки*
(повна назва)

Кафедра *Обчислювальної техніки*
(повна назва)

Освітньо-кваліфікаційний ступінь *магістр*
(назва ОКР)

Спеціальність *121. Інженерія програмного забезпечення*
(код і назва)

Спеціалізація *121. Інженерія програмного забезпечення комп'ютерних систем*

ЗАТВЕРДЖУЮ
Завідувач кафедри
С.Г. Стіренко
(підпис) (ініціали, прізвище)
« » 2020 р.

ЗАВДАННЯ
на магістерську дисертацію студенту
Ковбаса Святослав Ігорович
(прізвище, ім'я, по батькові)

1. Тема дисертації *Метод резервування та відновлення даних при їх зберіганні на віддалених носіях*

Науковий керівник дисертації *Марковський О. П., к. т. н., доцент*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від ***«26» 10 2020р. № 3132-с***

2. Строк подання студентом дисертації _____

3. Об'єкт дослідження процедури резервування та відновлення даних з віддалених носіїв зберігання інформації, доступ до яких втрачено

4. Предмет дослідження методи та засоби відновлення даних при втраченому доступі до одного-трьох носіїв в розподілених системах віддаленого зберігання інформації, які використовують теоретично найменшу кількість додаткових носіїв і дозволяють досягти високої продуктивності процедури відновлення даних

5. Перелік завдань, які потрібно розробити: Аналіз сучасного стану технологій розподіленого віддаленого зберігання інформації. Аналіз існуючих методів резервування та відновлення втрачених даних, виявлення

можливостей підвищення їх ефективності. Розробка та дослідження методу резервування та відновлення даних з двох втрачених дисків з використанням лінійних кодів. Розробка та дослідження модифікації методу резервування та відновлення даних з двох втрачених дисків з використанням діагональних сум. Розробка та дослідження покращення модифікації спрощеного методу резервування та відновлення даних з трьох втрачених дисків. Розробка програмного забезпечення для реалізації спрощеного методу резервування та відновлення даних із трьох втрачених носіїв на основі діагональних сум

6. Консультанти розділів дисертації:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Жабін В.І., проф., д.т.н		

7. Дата видачі завдання 26.12.2019 р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Строк виконання етапів дисертації	Примітка
1	Аналіз проблеми резервування та відновлення даних на віддалених носіях	24.02.2020	
2	Огляд літературних джерел	05.04.2020	
3	Розробка методу відновлення двох втрачених носіїв з використанням лінійних кодів	30.04.2020	
4	Розробка модифікації методу відновлення двох втрачених носіїв	28.05.2020	
5	Розробка спрощеного методу відновлення трьох втрачених носіїв	04.09.2020	
6	Розробка програмних засобів реалізації розробленого методу	28.09.2020	
7	Написання текстової частини	30.10.2020	
8	Оформлення дисертації	20.11.2020	

Студент _____ *С. І. Ковбаса*
(підпис) (ініціали, прізвище)

Науковий керівник дисертації _____ *О. П. Марковський*
(підпис) (ініціали, прізвище)

РЕФЕРАТ

на магістерську дисертацію

виконану на тему: Метод резервування та відновлення даних при їх зберіганні на віддалених носіях

студентом: Ковбасою Святославом Ігоровичем

Робота складається із вступу та чотирьох розділів. Загальний обсяг роботи: 90 аркушів основного тексту, 20 ілюстрацій, 22 таблиці. При підготовці використовувалася література із 41 різного джерела.

Актуальність теми. Особливостями сучасного етапу вдосконалення інформаційних технологій є поглиблення процесів інформаційної інтеграції та значне розширення застосування хмарних та розподілених обчислень.

Прогрес технології глобальних мереж, досягнутий в останні десять років, докорінно змінив організацію комп’ютерної обробки інформації на користь розподілених систем зберігання даних та віддалених обчислень. Широкого розповсюдження набула практика розподіленого зберігання інформації користувачів на віддалених сховищах. Крім таких очевидних переваг, як практична відсутність обмежень на об’єм пам'яті, доступність інформації користувача для широкого кола інших користувачів, зручність обміну даними, сучасна технологія зберігання інформації в хмарах забезпечує якісно більш високий рівень надійності.

Втрата даних при їх зберіганні на магнітних, оптичних чи твердотільних накопичувачах може відбуватися в результаті виходу з ладу обладнання, помилок в роботі програмного забезпечення, хакерських атак, дій вірусних програм, комп’ютерних хробаків, катаклізмів політичного, терористичного, природного, військового чи техногенного характеру. При цьому можуть втрачатися як окремі блоки даних, так і вся інформація на певному сховищі. При зберіганні даних користувачів у хмарі, основну роль в забезпеченні

надійності відіграє рознесення даних користувача по різним, географічно віддаленим, сховищам. Це обмежує втрати даних під дією наведених вище причин, відносно невеликим об’ємом. Втрачені дані можуть бути відновлені за рахунок використання резервних блоків, які формуються з інформаційних блоків.

В цьому ракурсі резервування та відновлення втрачених даних відіграють важливу роль в технології забезпечення високої надійності розподіленого зберігання інформації користувачів. Відповідно, подальший розвиток технологій віддаленого зберігання даних диктує необхідність адекватного вдосконалення методів їх резервування та відновлення втраченої інформації.

Таким чином, наукова задача підвищення ефективності резервування та відновлення даних користувачів при їх розподіленому зберіганні на віддалених сховищах є актуальною для сучасного етапу розвитку інформаційних та комп’ютерних технологій.

Мета і задачі дослідження. Метою цієї роботи є підвищення ефективності відновлення даних, втрачених при їх віддаленому зберіганні за рахунок розробки методів резервування, які враховують особливості реальних систем віддаленого інформації на розподілених сховищах, а також дозволяють прискорити процес відновлення даних, доступ до яких втрачено. Задачами дослідження, відповідно до поставленої мети є:

- Дослідження та аналіз можливостей застосування лінійних кодів для відновлення втрачених даних на дисках.
- Розробка та дослідження методу відновлення даних з двох втрачених дисків з використанням двох додаткових дисків та трьох втрачених дисків з використанням трьох додаткових дисків на основі діагональних та прямих контрольних сум, а також розробка покращень, що дозволяють досягти високої продуктивності відновлення даних при програмній та апаратній реалізації.

- Розробка програмного забезпечення для реалізації спрощеного методу резервування та відновлення даних із трьох втрачених носіїв на основі діагональних та прямих сум, які дозволяють прискорити процес відновлення даних із втрачених дисків.

Об’єкт дослідження — процедури резервування та відновлення даних з віддалених носіїв зберігання інформації, доступ до яких втрачено.

Предмет дослідження — методи та засоби відновлення даних при втраченому доступі до одного-трьох носіїв в розподілених системах віддаленого зберігання інформації, які використовують теоретично найменшу кількість додаткових носіїв і дозволяють досягти високої продуктивності процедури відновлення даних.

Методи дослідження. Для досягнення поставлених в магістерській роботі задач, використано теорію ймовірності та математичної статистики, теорію організації обчислень, теорію комбінаторики та булевих функцій, а також застосування методів моделювання систем

Наукова новизна одержаних результатів полягає в наступному:

- Запропоновано та досліджено метод прискореного відновлення обмеженої кількості втрачених інформаційних блоків, який відрізняється тим, що кількість блоків даних, з яких формується кожен з резервних блоків обмежена заданими граничними значеннями часу реконструкції втрачених блоків, що дозволяє за рахунок збільшення надлишковості резервування, прискорити процес відновлення інформації при її віддаленому зберіганні в порівнянні з відомими методами.

- Запропоновано модифікацію методу відновлення даних з двох втрачених дисків з використанням двох додаткових дисків, а також трьох втрачених дисків із використанням трьох додаткових, яка полягає в формуванні кодів другого та третього додаткових накопичувачів у вигляді сум за модулем 2 всіх основних та побічних діагоналей матриці. Доведено, що в запропонованій модифікації кількість слів на другому та третьому додаткових накопичувачах становить $m+n-1 \approx m$. Суттєвою перевагою запропонованого способу є простота

обчислювальних процедур відновлення, які легко можуть бути реалізовані на апаратному рівні і забезпечують більшу продуктивність в порівнянні з іншими методами, які потребують розв’язання систем рівнянь.

Практичне значення отриманих результатів роботи визначається тим, що використання розроблених методів дозволяє забезпечити ефективне резервування інформації, що зберігається в віддалених рознесених сховищах, з урахуванням вимог до надійності її доступності кожного користувача. Отримані результати дозволяють забезпечити заданий рівень доступності даних конкретного користувача відповідно до його вимог щодо надійності та оперативності доступу до даних, що зберігаються на віддалених від нього носіях. Отримані результати дозволяють за рахунок використання простих в обчислювальній реалізації лінійних перетворень та оптимізованих специфікацій значно прискорити процес відновлення втрачених даних.

Особистий внесок здобувача. Магістерська дисертація є самостійно виконаною роботою, в якій відображено особистий авторський підхід та особисто отримані теоретичні та прикладні результати, що відносяться до вирішення задачі резервування та відновлення даних при їх зберіганні на віддалених носіях. Формулювання мети та завдань дослідження проводилось спільно з науковим керівником.

Ключові слова

резервування даних, відновлення даних, backup, віддалені носії, надійність зберігання даних, виправлення помилок.

ЗМІСТ

ВСТУП

Актуальність теми. Особливостями сучасного етапу вдосконалення інформаційних технологій є поглиблення процесів інформаційної інтеграції та значне розширення застосування хмарних та розподілених обчислень.

Прогрес технології глобальних мереж, досягнутий в останні десять років, докорінно змінив організацію комп’ютерної обробки інформації на користь розподілених систем зберігання даних та віддалених обчислень. Широкого розповсюдження набула практика розподіленого зберігання інформації користувачів на віддалених сховищах. Крім таких очевидних переваг, як практична відсутність обмежень на об’єм пам'яті, доступність інформації користувача для широкого кола інших користувачів, зручність обміну даними, сучасна технологія зберігання інформації в хмарах забезпечує якісно більш високий рівень надійності.

Втрата даних при їх зберіганні на магнітних, оптичних чи твердотільних накопичувачах може відбуватися в результаті виходу з ладу обладнання, помилок в роботі програмного забезпечення, хакерських атак, дій вірусних програм, комп’ютерних хробаків, катаклізмів політичного, терористичного, природного, військового чи техногенного характеру. При цьому можуть втрачатися як окремі блоки даних, так і вся інформація на певному сховищі. При зберіганні даних користувачів у хмарі, основну роль в забезпеченні надійності відіграє рознесення даних користувача по різним, географічно віддаленим, сховищам. Це обмежує втрати даних під дією наведених вище причин, відносно невеликим об’ємом. Втрачені дані можуть бути відновлені за рахунок використання резервних блоків, які формуються з інформаційних блоків.

В цьому ракурсі резервування та відновлення втрачених даних відіграють важливу роль в технології забезпечення високої надійності розподіленого зберігання інформації користувачів. Відповідно, подальший розвиток

технологій віддаленого зберігання даних диктує необхідність адекватного вдосконалення методів їх резервування та відновлення втраченої інформації.

Таким чином, наукова задача підвищення ефективності резервування та відновлення даних користувачів при їх розподіленому зберіганні на віддалених сховищах є актуальною для сучасного етапу розвитку інформаційних та комп’ютерних технологій.

Мета і завдання дослідження. Метою цієї роботи є підвищення ефективності відновлення даних, втрачених при їх віддаленому зберіганні за рахунок розробки методів резервування, які враховують особливості реальних систем віддаленого інформації на розподілених сховищах, а також дозволяють прискорити процес відновлення даних, доступ до яких втрачено.

Об’єктом дослідження є процедури резервування та відновлення даних з віддалених носіїв зберігання інформації, доступ до яких втрачено.

Предметом дослідження є методи та засоби відновлення даних при втраченому доступі до одного-трьох носіїв в розподілених системах віддаленого зберігання інформації, які використовують теоретично найменшу кількість додаткових носіїв і дозволяють досягти високої продуктивності процедури відновлення даних.

Основними задачами дослідження, відповідно до поставленої мети, є:

- Аналіз та огляд сучасного та перспективного стану технологій розподіленого віддаленого зберігання інформації, обґрунтування критеріїв ефективності засобів резервування даних в відданих сховищах та при передачі по мережам. Критичний огляд, з позицій визначених критеріїв, існуючих методів резервування та відновлення втрачених даних, виявлення можливостей підвищення їх ефективності, а також визначення напрямків досліджень по імплементації цих можливостей.
- Дослідження та аналіз можливостей застосування лінійних кодів для відновлення втрачених даних на дисках.
- Розробка та дослідження методу відновлення даних з двох втрачених дисків з використанням двох додаткових дисків та трьох втрачених дисків з

використанням трьох додаткових дисків на основі діагональних та прямих контрольних сум, а також розробка покращень, що дозволяють досягти високої продуктивності відновлення даних при програмній та апаратній реалізації.

- Розробка програмного забезпечення для реалізації спрощеного методу резервування та відновлення даних із трьох втрачених носіїв на основі діагональних та прямих сум, які дозволяють прискорити процес відновлення даних із втрачених дисків.

Методи дослідження базуються на основі теорії ймовірності та математичної статистики, теорії організації обчислень, теорії комбінаторики та булевих функцій, а також на застосуванні методів моделювання систем.

Наукова новизна одержаних результатів полягає в наступному:

- Запропоновано та досліджено метод прискореного відновлення обмеженої кількості втрачених інформаційних блоків, який відрізняється тим, що кількість блоків даних, з яких формується кожен з резервних блоків обмежена заданими граничними значеннями часу реконструкції втрачених блоків, що дозволяє за рахунок збільшення надлишковості резервування, прискорити процес відновлення інформації при її віддаленому зберіганні в порівнянні з відомими методами.

- Запропоновано модифікацію методу відновлення даних з двох втрачених дисків з використанням двох додаткових дисків, а також трьох втрачених дисків із використанням трьох додаткових, яка полягає в формуванні кодів другого та третього додаткових накопичувачів у вигляді сум за модулем 2 всіх основних та побічних діагоналей матриці. Доведено, що в запропонованій модифікації кількість слів на другому та третьому додаткових накопичувачах становить $m+n-1 \approx m$. Суттєвою перевагою запропонованого способу є простота обчислювальних процедур відновлення, які легко можуть бути реалізовані на апаратному рівні і забезпечують більшу продуктивність в порівнянні з іншими методами, які потребують розв’язання систем рівнянь.

Практична цінність отриманих результатів роботи визначається тим, що використання розроблених методів дозволяє забезпечити ефективне

резервування інформації, що зберігається в віддалених рознесених сховищах, з урахуванням вимог до надійності її доступності кожного користувача. Отримані результати дозволяють забезпечити заданий рівень доступності даних конкретного користувача відповідно до його вимог щодо надійності та оперативності доступу до даних, що зберігаються на віддалених від нього носіях. Отримані результати дозволяють за рахунок використання простих в обчислювальній реалізації лінійних перетворень та оптимізованих специфікацій значно прискорити процес відновлення втрачених даних.

РОЗДІЛ 1
ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

1.1. Огляд існуючих способів зберігання даних на віддалених носіях

Сьогодні існує кілька різних способів побудови сховищ даних. Найбільш популярними із них є технології DAS (Direct-Attached Storage, носії із прямим підключенням) [1], NAS (Network-Attached Storage, носії із підключенням через мережу) [2] та SAN (Storage Area Network, мережа зберігання даних).

Система DAS підключається напряму до сервера, без мережевих вузлів між носієм та сервером. Для того, аби інші сервери мали доступ, вони звертаються до носія через підключений до нього сервер. Основною перевагою систем DAS є низька вартість та вища швидкість роботи носіїв, ніж по мережі. Проте такі системи мають обмеження, до яких входить розташування носіїв (вони повинні знаходитись безпосередньо біля сервера), що спричиняє також проблеми обмеженої масштабованості [3]. Таким чином для організації віддаленого доступу до таких носіїв, сервер повинен сам надавати мережевий інтерфейс для роботи із ним.

Системи SAN є розширенням моделі DAS. Вони пропонують доступ до носія по мережі, при цьому для операційної системи таке підключення виглядає точно так само, як і підключення напряму. Крім цього, особливістю цих систем є те, що вони надають доступ до носія на рівні блокового пристрою, що надає більшу гнучкість для клієнтів при виборі файлової системи. На відміну від DAS, для якої проблемами є обмежена масштабованість, складність управління масивом носіїв і можливі проблеми виникнення боттлнеків на серверах і локальних мережах, SAN надає засоби для автоматичного резервного копіювання, моніторингу за станами носіїв та процесом резервування та інші засоби для більш надійного зберігання інформації [4]. Більше того, у SAN є можливість збирати у собі велику кількість сховищ прозоро для клієнтів, що дає можливість легко масштабуватися та спрощує управління великою кількістю

носіїв. Таким чином, головними перевагами моделі SAN є висока продуктивність, масштабованість, централізація управління, ділимість між клієнтами та надійність у порівнянні з іншими системами зберігання даних на віддалених носіях [5]. Ще однією особливістю роботи SAN є спеціально розроблені низькорівневі мережеві протоколи, за допомогою яких виконується передача даних між клієнтом та мережею SAN. Основними таким протоколами є Fibre Channel та iSCSI. Fibre Channel був фактичним стандартом при зародженні SAN систем, проте потребує окремої лінії для користування. iSCSI є більш сучасним протоколом та може працювати поверх загальної мережі Internet.

Отже, до переваг SAN можна віднести:

- Високий рівень гнучкості;
- Висока рівень відмовостійкості;
- Централізоване управління;
- Незалежність топології SAN від серверів;
- Високий рівень швидкодії;
- Вони легко масштабуються;

До недоліків SAN відносяться:

- Велика ціна та складність установки;

Системи NAS, як і SAN, надають клієнтам доступ до носіїв по мережі, проте на рівні файлової системи. Доступ до NAS в основному представляється через виділений сервер. Працює ця система за такими протоколами, як NFS та SMB. Перевагою доступу на рівні файлової системи є те, що клієнти є абстрагованими від низькорівневої структури диску, що інколи є бажаним ефектом. З іншого боку, інколи доцільно мати доступ саме на рівні блочного пристрою, тому це одночасно є й недоліком NAS [6].

Особливостями NAS є:

- Простота встановлення;
- Робота на рівні файлової системи - через це доступ до даних не залежить від операційної системи та платформи;

- Низька масштабованість — доступ до носіїв відбувається через виділений сервер.

Враховуючи невисокий рівень масштабованості, протоколи передачі даних і нижчу вартість встановлення та підтримки, ніж SAN, зазвичай системи NAS використовуються для більш малобюджетних завдань.

Таким чином, описані системи збереження і доступу до даних по-різному вирішують завдання забезпечення надійності роботи, масштабованості та високої швидкості передачі даних.

Проте, найбільш поширеним методом підвищення надійності збережених даних є підвищення їх надлишковості. Хоча такий метод виправдовує себе для більшості завдань, його застосування для резервування та відновлення великих масивів даних пов'язане з деякими складнощами [7]. Наприклад, поломка одного чи декількох вузлів з носіями може призвести до складних проблем, що інколи і взагалі можуть не вирішуватися. Урахувавши специфіку зберігання великих даних, можна сказати, що розглянуті системи зберігання і доступу до даних не забезпечують самі по собі в повній мірі необхідного рівня надійності та продуктивності.

1.2. Огляд існуючих способів резервування та відновлення помилок

На сьогодні вже існує певна кількість методів резервування та відновлення даних на віддалених носіях [8-11].

Для систем резервування найважливішими критеріями є:

- Кількість носіїв, з яких, у разі втрати доступу, дані можуть бути відновлені;
- Складність обчислень процедури відновлення та резервування даних;
- Коефіцієнт-відношення кількості резервних носіїв до кількості носіїв, дані з яких можуть бути відновлені.

Найбільш розповсюдженими на практиці є технології резервування та відновлення, створені в рамках системи RAID (англ. Redundant Array of

Independent Disks — надлишковий масив незалежних дисків) [12]. Перші системи резервування під назвою RAID 1 створені на основі простого дзеркалювання даних, то більш розвинуті версії, включаючи RAID 5, передбачають використання корегуючих та стираючих кодів. Особливістю корегуючих кодів у порівнянні зі стираючими є те, що локація помилок відома апріорі.

RAID – це масив кількох фізичних незалежних дисків, які дозволяють досягти високої відмовостійкості та продуктивності. RAID підвищує ефективність операцій вводу-виводу, а також продуктивність і надійність носіїв. RAID може знаходитися на машині або у вигляді одного блоку зберігання даних, або декількох віртуальних пристроїв. Система дозволяє відновити втрачені дані шляхом їх реконструкції з решти дисків, при чому в залежності від рівня, засоби відновлення та зберігання даних міняються. Розглянемо різні рівні RAID.

RAID 0 (рис. 1.1) розподіляє дані рівномірно по двох або більше дисках, без інформації про парність, надлишковості або іншої стійкості до несправностей. Оскільки RAID 0 не забезпечує відмовостійкості або надлишковості даних, вихід з ладу одного диска призведе до виходу з ладу всього масиву; в результаті розміщення даних на всіх дисках, збій призведе до повної втрати даних. Ця конфігурація зазвичай використовується для підвищення продуктивності, хоча його також можна використовувати як спосіб створення великого логічного тому з двох або більше фізичних дисків [13].

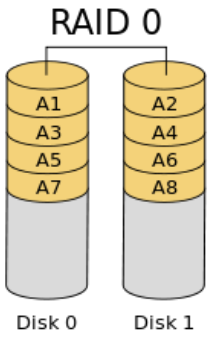

Рис. 1.1. Схема побудови RAID 0

RAID 1 (рис 1.2) складається з точної копії (або так званого дзеркала) набору даних на двох або більше дисках; класична дзеркальна пара RAID 1 містить два диски. Ця конфігурація не забезпечує парності, розділення (як у RAID 0) чи об’єднання декількох диків у один логічний, оскільки дані при запису відображаються на всі диски, що належать до масиву, і загальна ємність масиву буде дорівнювати ємності найменшого диска. Ця конфігурація корисна, коли продуктивність читання або надійність важливіші, ніж продуктивність запису або загальна ємність даних. Масив дисків буде продовжувати працювати до тих пір, поки працює хоча б один із його дисків [14].

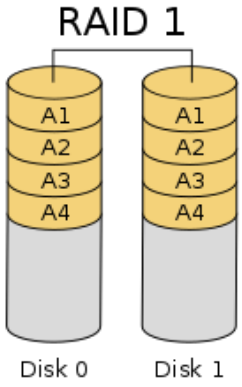

Рис. 1.2. Схема побудови RAID 1

RAID 2 (рис. 1.3), який доволі рідко використовується на практиці, ділить дані на дисках на рівні бітів і використовує код Хеммінга для виправлення помилок. Так як всі диски реалізують внутрішню корекцію помилок, складність зовнішнього коду Хеммінга мала незначну перевагу перед парністю, тому RAID 2 застосовується рідко [15].

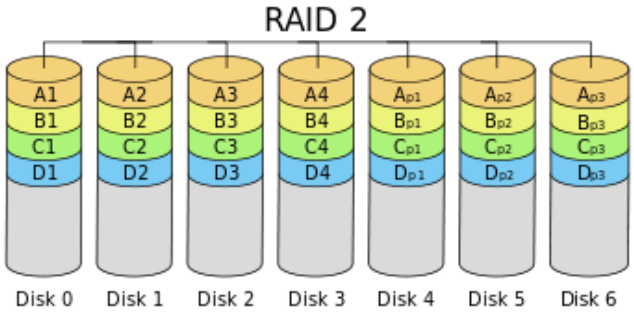

Рис. 1.3. Схема побудови RAID 2

RAID 3 (рис. 1.3), який також рідко використовується на практиці, складається із поділу дисків на рівні байтів із виділеним диском парності. Однією з характеристик RAID 3 є те, що він, як правило, не може обслуговувати декілька запитів одночасно, що спричинене тим, що будь-який окремий блок даних за визначенням розподіляється по всіх дисках і знаходиться на одній і тій же фізичній локації на кожному диску. Тому будь-яка операція введення-виведення вимагає активності на кожному диску і вимагає синхронізації [16].

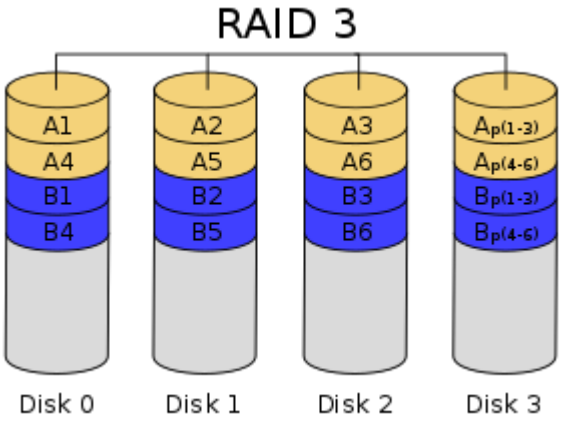

Рис. 1.3. Схема побудови RAID 3

RAID 4 (рис. 1.4) ділить диски на блоки із виділеним диском парності. В результаті своєї конструкції RAID 4 забезпечує хорошу продуктивність випадкових зчитувань, тоді як продуктивність випадкових записів є низькою через необхідність запису всіх даних про парність на один диск [17].

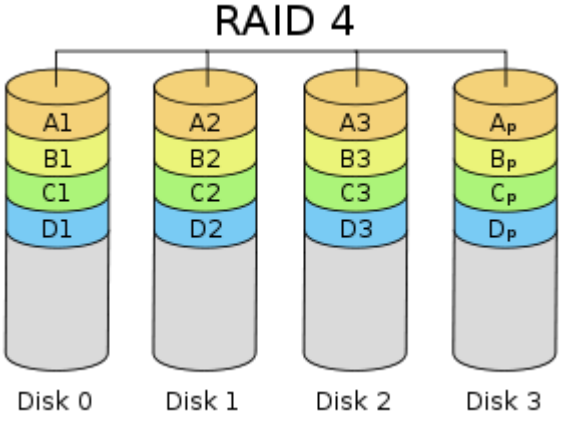

Рис. 1.4. Схема побудови RAID 4

RAID 5 (рис. 1.5) складається із поділу дисків на рівні блоку з розподіленою парністю. На відміну від RAID 4, інформація про парність розподіляється між усіма дисками. Це потрібно для того, щоб при зчитуванні та записі працювали всі носії, замість одного. При відмові одного із дисків, його дані можуть бути обчислені з розподіленої парності таким чином, що дані не втрачаються. Конструкція RAID 5 вимагає принаймні трьох дисків [18].

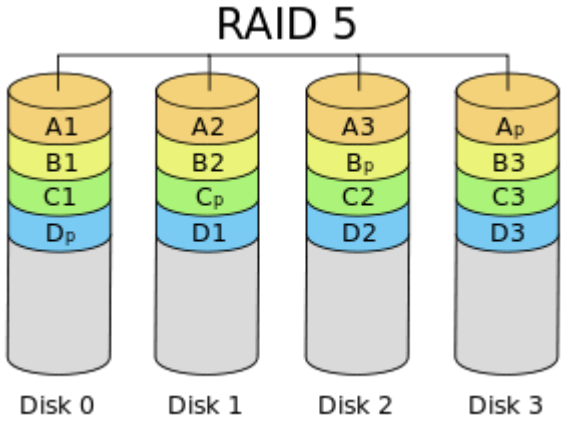

Рис. 1.5. Схема побудови RAID 5

RAID 6 (рис. 1.6) розширює RAID 5, додаючи ще один блок парності; таким чином, він використовує поділ диска на рівні блоку з двома блоками парності, розподіленими по всіх дисках [19-20].

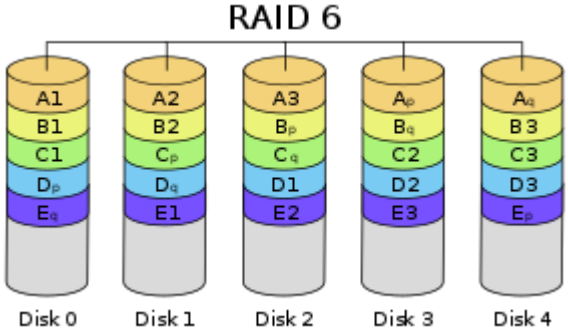

Рис. 1.6. Схема побудови RAID 6

Крім цього, існують комбіновані рівні RAID, такі як RAID 10, RAID 50 та ін. Ці рівні представляють із себе в основному комбінацію розподілу даних по дисках для швидшого читання та запису (RAID 0) та засоби забезпечення відмовостійкості, такі як дзеркалювання (RAID 10) та диски парності (RAID 50).

Окрім RAID існують й інші системи забезпечення резервування та відновлення помилок. Наприклад, MirrorFolder, що представляє із себе папку, яка дзеркалює та синхронізує встановлене ПЗ для резервного копіювання файлів на комп’ютері. Ця система надає широкий вибір параметрів синхронізації, зокрема синхронізацію в реальному часі, яка постійно синхронізує файли, та періодичний режим, який синхронізує файли із заданим періодом.

Ще одним прикладом системи для резервування даних є AnyFileBackup — це програма для моніторингу, резервування, синхронізації і шарингу різного виду файлів.

Прикладом використання саме коректуючих кодів є файлова система RozoFS, призначена саме для ефективного та надійного керування розподіленими носіями даних.

1.2. Огляд та аналіз існуючих способів корекції помилок

Історично виділяють два метода корекції помилок:

1. ARQ (англ. Automatic Repeat Request) — метод, при якому виконується автоматичний повтор передачі при виявленні помилок. Перевагою ARQ-методу є мала кількість контрольних розрядів та простота алгоритму виявлення помилки. При цьому для відновлення помилок при зберіганні даних ця технологія не є актуальною, так як при зберіганні даних повторні дані передати неможливо. За схожим принципом працює дзеркалювання — коли зберігається повна копія даних, і за детекції помилки ця копія даних відновлюється. Проте знову ж таки, через базову

реалізацію корекції помилок в самих пристроях, цю технологію не можна віднести до проблеми резервування та відновлення збережених даних.

2. FEC (англ. Forward Error Correction) — метод виправлення помилок “наперед”. До цього методу відносяться корегуючі коди, які дозволяють одразу виправити деяку кількість помилок за рахунок передобчислених “наперед” контрольних кодів, що зберігаються разом із даними [21];

Для цієї роботи розглядатися буде саме технологія FEC. До FEC, окрім звичайних корегуючих кодів належать також стираючі (erasure) коди. Особливістю стираючих кодів є те, що вони працюють під припущенням, що відновлювати доведеться саме втрачені (стерті) дані, а не пошкоджені. Іншими їх особливостями є те, що вони здатні виправляти цілі пакети даних у разі їх втрати, а також для них вважається, що позиції помилок уже відомі деяким іншим способом [22]. Їх перевагою є спрощення обчислень та можливість виправлення більшої кількості помилок у порівнянні з аналогічними корегуючими кодами.

Так як усі умови використання стираючих кодів виконуються для переважної кількості систем віддаленого зберігання даних, то саме вони є найефективнішими для резервування та відновлення даних і найчастіше використовуються саме для таких цілей [23].

Стираючі коди можна поділити на лінійні та циклічні. Недоліком лінійних кодів є високе значення надлишковості кода при відновленні даних із декількох носіїв. Алгоритм відновлення помилок у лінійних кодах представляє із себе розв’язання систем лінійних рівнянь [24]. Циклічні стираючі коди в основному є модифікацією кодів Ріда-Соломона. Вони будуються на основі полів Галуа і дозволяють досягти меншого рівня надлишковості, проте програють у продуктивності через високу часову складність процесу резервування та відновлення [25]. Розглянемо деякі популярні приклади стираючих кодів.

Код парності. Дані парності використовуються деякими RAID системами для досягнення надлишковості. Якщо диск у масиві виходить з ладу, його дані можна відновити із даних інших дисків (за допомогою операції XOR).

Наприклад, припустимо що, два диски в тридисковому масиві RAID 5 містять такі дані:

Диск 1: 10010110
Диск 2: 01010101

Для обчислення даних парності для двох накопичувачів, між їх даними виконується операція XOR:

```
  10010110
XOR 01010101
————————————
  11000011
```

Отримані дані про парність (1100001) зберігаються на диску 3.

Якщо будь-який з трьох носіїв виходить з ладу, вміст несправного накопичувача може бути відновлений із резервного, піддавши дані з інших накопичувачів тій самій операції XOR. Тобто, якщо диск 2 зазнає збою, його дані можуть бути відновлені за допомогою операції XOR між даними двох інших дисків, 1 та 3:

Диск 1: 10010110
Диск 3: 11000011

Диск 2:

```
  10010110
XOR 11000011
————————————
  01010101
```

Як бачимо, результат цього обчислення дає оригінальні дані на диску 2. Після обчислень, знайдені оригінальні дані знову зберігаються на диску 2, повністю відновлюючи його. Ця ж концепція застосовується і до більших масивів із використанням будь-якої кількості дисків. У випадку масиву RAID 3 з 12 накопичувачів, 11 накопичувачів беруть участь у показаному вище обчисленні XOR і дають значення, яке потім зберігається на виділеному 12-ому накопичувачі парності. Розширення механізму партості, такі як "подвійна" або "діагональна" парність використовуються в RAID-DP [26].

В загальному випадку, коли всього є *n* дисків із користувацькими даними, та 1 диск із даними парності, механізм коду парності описується формулами (1.1) та (1.2).

$$D_{k+1} = \bigoplus_{i=1}^{k} D_i \qquad (1.1)$$

$$D_e = \bigoplus_{i=1,\ i \neq e}^{k+1} D_i \qquad (1.2)$$

де k - кількість носіїв із даними;

D_i - i-тий носій з даними;

$k+1$ - індекс носія, на якому зберігаються дані парності;

e - індекс носія, на якому були втрачені дані.

Недоліком таких кодів є можливість відновлення лише одного носія, якщо два носія із системи вийдуть з ладу в один період часу, то такі коди не допоможуть відновити ніяку інформацію.

Іншими дуже поширеними стираючими кодами, які можуть виступати і як коректуючі, є коди Ріда-Соломона [27]. Коди Ріда-Соломона входять у категорію недвійкових циклічних кодів, які можуть виправляти довільну кількість спотворених символів з використанням вдвічі більшої кількості контрольних символів, або довільну кількість втрачених (стертих) символів з використанням такої ж кількості контрольних символів. Якщо вважати, що q — це розрядність

одного символа в повідомленні, то загальний розмір блоку для кодів Ріда-Соломона *n* є обмеженим і пов'язаний із *m* співвідношенням (1.3).

$$n \leq 2^q - 1 \qquad (1.3)$$

Для генерування корегуючого коду Ріда-Соломона потрібно взяти простий поліном на полях Галуа степені *k* — кількості контрольних символів. Такий поліном називається генеруючим, позначається *g(x)* і будується так, щоб його коренями були коефіцієнти $\alpha, \alpha^2, \alpha^3, ..., \alpha^k$, що є символами заданого поля Галуа. Формується він за формулою (1.4).

$$g(x) = (x - \alpha)\left(x - \alpha^2\right)..(x - \alpha^k) \qquad (1.4)$$

Тоді, для формування контрольних кодів, інформаційні символи повідомлення перетворюють у відповідний йому поліном у заданому полі, домножають його на x^k (відповідає зсуву вліво на *k* розрядів) та ділять на генеруючий поліном. Поліном, що є остачею від цього ділення і є при переведенні назад у двійкову систему контрольними символами (1.5).

$$R(x) = m(x) \cdot x^k + m(x) \bmod g(x) \qquad (1.5)$$

де *R(x)* - результуюче повідомлення з інформаційними та контрольними символами, яке здатне виправляти до *k / 2* помилок.

Очевидним є те, що *R(x)* ділиться націло на *g(x)*, тобто коефіцієнти $\alpha, \alpha^2, \alpha^3, ..., \alpha^k$ є коренями рівняння *R(x)*. Значення $S_j = R(\alpha^j)$ називаються синдромами помилок, і якщо хоча б один із них при читанні повідомлення не дорівнює нулю, це означає, що у прийнятому повідомленні є помилка. Позначивши через X_k^j локацію *j*-тої помилки у *k*-тому синдромі, а через Y_k значення *k*-тої помилки, виправлення помилок зводиться до розв'язання системи рівнянь (1.6).

$$\begin{bmatrix} X_1^1 & X_2^1 & \cdots & X_\nu^1 \\ X_1^2 & X_2^2 & \cdots & X_\nu^2 \\ \vdots & \vdots & & \vdots \\ X_1^{n-k} & X_2^{n-k} & \cdots & X_\nu^{n-k} \end{bmatrix} \begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_\nu \end{bmatrix} = \begin{bmatrix} S_1 \\ S_2 \\ \vdots \\ S_{n-k} \end{bmatrix} \qquad (1.6)$$

При цьому, при використанні кодів Ріда-Соломона як стираючих, вважається, що позиції вже відомі, і на цьому етапі обчислення припиняються — розв’язок цієї системи можна одразу отримати, і він дозволяє отримати оригінальні значення втрачених даних [28].

З іншого боку, якщо коди Ріда-Соломона використовуються як коректуючі, то для корекції помилок необхідно ще знайти позиції помилок X_k^j, для чого потрібно скористатись поліномом локатора помилок (1.7).

$$\Lambda(x) = \prod_{k=1}^{\nu}(1 - xX_k) = 1 + \Lambda_1 x^1 + \Lambda_2 x^2 + \cdots + \Lambda_\nu x^\nu \qquad (1.7)$$

Великим недоліком кодів Ріда-Соломона є висока обчислювальна складність [29]. Спричинена вона складністю виконання певних операцій на полях Галуа, зокрема множенню на полях Галуа, яке не імплементоване апаратно у процесорах. Програмна реалізація множення має виконуватися шляхом послідовного виконання операцій зсуву та XOR.

При декодуванні виконуються дві операції — виявлення помилок та їх виправлення. При виявленні обчислюється значення *k* синдромів. Якщо всі їх значення дорівнюють нулю, це говорить про те, що зчитане повідомлення не містить помилок. При обчисленні кожного з синдромів проводиться n операцій множення і така ж кількість операцій додавання. Тому, часова складність виявлення помилок T_1 розраховується як:

$$T_1 = k \cdot n \cdot \left(t_{MUL} + t_{XOR} \right) \qquad (1.8)$$

де t_{MUL} — час множення на полях Галуа, t_{XOR} — час виконання XOR. На полях Галуа операція множення виконується *m* ітерацій, на кожній з яких виконуються операції зсуву множеного і множника, аналізу (*q+1*)-го розряду множеного після його зсуву, тоді, якщо цей розряд дорівнює 1, то XOR із

константою, тоді аналіз значення поточного розряду множника і, якщо цей розряд є одиницею — ще одна операція XOR.

Отже, на кожній ітерації множення в середньому виконується дві операції зсуву і одна операція XOR, тобто $t_{MUL} = q \cdot (2t_S + t_{XOR})$. Звідси слідує те, що загальна обчислювальна складність операції виявлення помилок становить $O(4qkn)$.

Обчислення k синдромів може бути здійснене паралельно. Однак обчислення одного синдрому є послідовним, тобто обчислювальна складність контролю присутності помилок дорівнює $O(4nm)$. Обчислювальна складність дій, пов'язаних з корекцією помилок становить $O(nmk)$. Таким чином, витрати ресурсів процесора на виконання процедур локалізації та корекції помилок є значними, що пояснюється виконанням множення на полях Галуа.

З корегуючих кодів, що орієнтовані на виправлення бітових помилок одними з найбільш відомих є коди Хеммінга, вже розглянуті коди Ріда-Соломона, а також коди на основі зважених сум. Корегуючі коди діляться на дві групи — неперервні та блочні [30].

Неперервні коди складаються із неперервної послідовності символів, яка на блоки не ділиться.

Блочні коди складаються із окремих блоків кодових комбінацій, які незалежно одне від одного декодуються та кодуються.

Блочні коди, у свою чергу, враховуючи характер перетворень, що в них відбуваються, ділять на нелінійні та лінійні. Лінійний блочний код — це код, всі кодові слова якого формують лінійний простір векторів. Це означає, що будь-яку нову не заборонену комбінацію коду можливо отримати через лінійну комбінацію його інших не заборонених кодів. Відповідно, решту блочних кодів, які не піддаються цим правилам називають нелінійними [31].

Також існують блочні систематичні коди, що є (n,k)-кодами, у яких кодові слова формуються із k інформаційних символів та n-k контрольних символів. При цьому, ще однією властивістю цих кодів є те, що позиції інформаційних символів та контрольних символів є одними й тими ж для всіх слів коду. Така

структура кодів значно спрощує реалізацію кодування та декодування, через це такі коди є надзвичайно розповсюдженими [32]. До систематичних кодів відносяться коди Хеммінга та усі лінійні циклічні коди, такі як БЧХ [33].

Одним із перших корегуючих кодів є код Хеммінга. Код Хеммінга дозволяє виявляти помилки *n*-розрядного блоку даних $D = \{d_1, d_2, \ldots, d_n\}$. Для цього з допомогою цього кода формуються *m* контрольних сум $r_1, r_2, \ldots, r_m$ для різних груп розрядів $C_1, C_2, \ldots, C_m$ бітів блоку *D (1.9)*:

$$\begin{aligned} &\forall i \in \{1, \ldots, n\} : d_j \in C_j \Rightarrow b_{ji} = 1, d_j \notin C_j \Rightarrow b_{ji} = 0 \\ &\forall j \in \{1, \ldots, m\} : r_j = b_{j1} d_1 \oplus b_{j2} d_2 \oplus \ldots \oplus b_{jn} d_n \end{aligned} \qquad (1.9)$$

У ролі контрольних розрядів коду виступають саме ці контрольні суми. Вони вираховуються при зберіганні даних і зберігаються разом із блоком даних *D*. При читанні повідомлення, контрольні розряди знову обчислюються і порівнюються зі збереженими. Якщо блок був переданий без помилок, всі контрольні розряди будуть співпадати, інакше вважається, що виникла помилка [34]. Коди Хеммінга здатні корегувати однократні помилки, а виявляти дві (при додаванні додаткового біта парності).

Більшу кількість помилок дозволяють корегувати коди БЧХ, до яких відносяться і вже розглянуті коди Ріда-Соломона [35]. Ці коди базуються на основі арифметики Галуа і дозволяють виправляти велику кількість помилок з мінімумом надлишковості.

Проте застосування кодів БЧХ та Хемінга до блоків великої довжини є неефективним. До того ж, архітектура сучасних процесорів не пристосована до виконання операцій на полях Галуа.

Доволі ефективним та простим способом виправлення однократних помилок є метод зважених контрольних сум [36]. Нехай передається *n*-розрядний блок даних $D = \{d_1, d_2, \ldots, d_n\}$., тоді зважена контрольна сума *W*

вираховується як XOR добутків значень розрядів блоку на ваговий коефіцієнт цих розрядів, який фактично є їхнім порядковим номером (1.10).

$$C = \bigoplus_{i=1}^{n} d_i \cdot i \tag{1.10}$$

Через те, що розрядність індекса *i* становить $log_2(n+1)$, кількість контрольних бітів є такою ж. Контрольна сума обчислюється як при збереженні, так і при читанні даних, при цьому при читанні додатково обчислюється сума за модулем 2 між збереженими та обчисленими даними. Проблемою є те, що зважені контрольні суми не дають можливості для виправлення багатократних помилок. Звідси видно, що зважені контрольні суми, через свої особливості, погано підходять для резервування та відновлення даних із віддалених носіїв.

Висновки до розділу 1

У результаті виконаного огляду та аналізу існуючих рішень резервування та відновлення даних на віддалених носіях, а також огляду предметної області теми поточної роботи можна зробити наступні висновки:

1. За умов сучасного розвитку інформаційних технологій, великого поширення набуває використання розподілених та віддалених систем зберігання даних, для побудови яких існують декілька загальновідомих технологій, таких як SAN та NAS. Проте, ці технології самі по собі не забезпечують системи віддалених носіїв задовільним рівнем надійності. Через це з’являється потреба у додатковій реалізації резервування та відновлення даних для віддалених носіїв.

2. Основними критеріями ефективності методів резервування та відновлення даних є складність виконання обчислень для відповідних процедур, надлишковість резервних даних та кількість носіїв, що можуть бути відновлені. Аналіз існуючих методів резервування та відновлення даних показав, що вони або надають продуктивний метод для відновлення лише одного носія системи (RAID1, RAID3, RAID5), або дозволяють відновлювати два носія, проте використовують складні для обчислень операції, такі як множення на полях Галуа (RAID6). Звідси випливає, що є сенс розробляти метод резервування та відновлення даних, який буде дозволяти відновлювати більше одного втраченого носія, та при цьому, ефективно з точки зору часу обчислень, виконувати операції відновлення та резервування даних.

3. Для відновлення даних із втрачених носіїв, поширеним є використання корегуючих кодів. Для цих корегуючих кодів, що називають стираючими, ставляться дещо інші вимоги, ніж для корегуючих кодів, що використовуються при виправленні помилок певної кратності. Таким чином, для стираючих кодів не потрібно виконувати локалізацію помилок, так як вважається, що їх позиції відомі апріорі, дані вважаються втраченими, а не просто пошкодженими та оперують ці коди над певними блоками даних, що в

предметній області поточної роботи виступають носіям. Спрощені вимоги до цих кодів дозволяють розробляти більш ефективні методи. Незважаючи на це, існуючі коди для резервування та відновлення помилок мають недоліки. Таким чином для деяких кодів операції резервування та відновлення виконуються швидко (коди парності), проте вони здатні виправляти лише один блок даних. Інші коди, такі як коди Ріда-Соломона, хоча і дозволяються виправляти велику кількість помилок, проте є неефективними з точки зору обчислень. Це все підводить до необхідності створення корегуючих кодів, які дозволятимуть відновлювати декілька носіїв без просідання у швидкості їх роботи.

РОЗДІЛ 2
РОЗРОБКА МЕТОДІВ РЕЗЕРВУВАННЯ ТА ВІДНОВЛЕННЯ ДАНИХ ІЗ КІЛЬКОХ НОСІЇВ

При зберіганні інформації на віддалених носіях існує деяка ймовірність того, що на одному чи кількох носіях даних із деяких причин станеться втрата даних. Щоб користувач при цьому не втратив збережену інформацію, потрібно здійснювати її резервування із використанням додаткових віддалених носіїв. Враховуючи те, що у переважній більшості випадків ймовірність втрати даних на віддалених носіях несе біноміальний розподіл [37], ймовірність втрати даних одразу на декількох носіях стає в порядки меншою зі збільшенням кількості відмовлених носіїв, проте це не виключає можливості виникнення таких багатократних відмов.

2.1 Опис способу резервування та відновлення даних з двох носіїв із використанням лінійних кодів

Для того, аби забезпечити можливість виконання паралельних операцій зчитування та запису, а також уникнення локалізації даних на одному носії, пропонується дані, що записуються у вигляді файлу, розділити на n блоків, кожен із яких записується на відповідний віддалений носій, кількість яких також дорівнює n. Таким чином i-тий блок буде збережено на i-тому носії, де $i \in \{1,...,n\}$. Відповідно, на кожному такому i-тому носії буде всього збережено m блоків. Позначимо один j-тий блок з i-того носія як a_{ij}, де $i \in \{1,...,n\}$; $j \in \{1,...,m\}$. Припустимо, що були втрачені дані на носіях u та v, причому $u,v \in \{1,...,n\}$. Тоді, при втраті цих даних, потрібно відновити всі блоки a_{uj} та a_{vj}, де $j=1,...,m$. Причому для цього можна використовувати дані із решти $n-2$ загальних носіїв та r додаткових, у які при збереженні даних були записані певні контрольні суми. Для відновлення даних при їх втраті на одному носії пропонується

використати дуже поширений метод — зберігати на додаткових носіях суми за модулем 2 усіх відповідних блоків загальних носіїв:

$$\forall j = 1, \ldots, m: s_{1j} = \bigoplus_{i=1}^{n} a_{ij} \tag{2.1}$$

Якщо ж розглядати проблему відновлення даних, які зберігалися на двох і більше носіях, то для вирішення такої задачі потрібно розв'язувати системи рівнянь. Наприклад, при втраті даних із двох носіїв із номерами *u* та *v*, значення втрачених блоків можуть бути знайдені в результаті розв'язання системи двох лінійних булевих рівнянь наступного виду:

$$\begin{cases} a_{uj} \oplus a_{vj} = z_{1j} \\ a_{uj} = z_{2j} \end{cases}, \ \forall j \in \{1, \ldots, m\} \tag{2.2}$$

Або розв'язання тотожної системи:

$$\begin{cases} a_{uj} \oplus a_{vj} = z_{1j} \\ a_{vj} = z_{3j} \end{cases}, \ \forall j \in \{1, \ldots, m\} \tag{2.3}$$

Тут z_{kj} —лінійна комбінація контрольної суми, що зберігається на *k*-тому додатковому носії та даних, що зберігаються на загальних дисках.

У системах (2.2) та (2.3) перше рівняння виводиться із рівняння (2.1) за умови, що при обчисленні першої контрольної суми враховуються всі носії системи, та всі носії системи, крім носіїв під номерами *u* та *v* не є пошкодженими. Отримання другого рівняння із систем (2.2) та (2.3) із мінімальним використанням надлишковості є значно складнішою задачею, оскільки на момент обчислення контрольних сум значення *u* та *v* є невідомими. З іншого боку, очевидним є той факт, що це друге рівняння може бути отримане із системи булевих рівнянь, що при будь-яких можливих значеннях *u* та *v* містить деяке рівняння, в яке входять блоки тільки одного із носіїв *u* або *v*. Тобто в цьому рівнянні беруть участь лише блоки a_{uj} та a_{vj}, де $j=1,...,m$. Хорошим прикладом такої системи при кількості носіїв, що є степенем 2 (чи, іншими словами таке *n*, що $log_2 n \in Z$) є наступна система:

$$\begin{cases} a_{1j} + a_{2j} + \ldots + a_{m/2,j} = y_j \\ a_{1j} + a_{2j} + \ldots + a_{m/4,j} + a_{m/2+1,j} + \ldots + a_{3m/4,j} = y_2 \\ \ldots \\ a_{1j} + a_{3j} + a_{5j} + \ldots + a_{m-1,j} = y_{\log_2 n,j} \end{cases}, \forall j \in \{1,\ldots,m\} \quad (2.4)$$

де y_{kj} — контрольна сума для j-го блока, що зберігається на k-тому додатковому носії.

У цьому випадку система містить $log_2 n$ рівнянь. В загальному випадку, для довільного значення кількості носіїв, кількість рівнянь у такій системі буде дорівнювати $\lceil log_2 n \rceil$. Варто зауважити, що ця система рівнянь не включає останній n-тий носій, а також для формування систем (2.2) та (2.3) потрібно зберігати контрольну суму парності, що розраховується за рівнянням (2.1). Таким чином, контрольні суми, обчислені за системою (2.4) зберігаються на $\lceil log_2 n \rceil$ носіях, контрольна сума (2.1) зберігається на одному носієві та повне віддзеркалення n-го носія зберігається також на одному носієві, отже загальна кількість додаткових носіїв r, що є надлишковістю системи, вираховується за формулою:

$$r = 2 + \lceil \log_2 n \rceil \quad (2.5)$$

Для прикладу, якщо кількість загальних носіїв n становить 4, то аби мати змогу відновлювати втрачені дані із будь-яких двох відсутніх для доступу віддалених носіїв, потрібно додати $r=2+\lceil log_2 n \rceil=2+2=4$ додаткових носія, в яких повинні зберігатися всі m блоків, що формуються за наступною системою рівнянь:

$$\begin{cases} s_{1,j} = a_{1j} + a_{2j} + a_{3j} + a_{4j} \\ s_{2j} = a_{1j} + a_{2j} \\ s_{3j} = a_{1j} + a_{3j} \\ s_{4j} = a_{4j} \end{cases}, \forall j = 1,\ldots,m \quad (2.6)$$

де s_{kj} — контрольна сума j-го блока, що зберігається на k-тому додатковому носії.

Очевидним є те, що дана система є лінійно незалежною, і тому при будь-яких індексах пошкоджених носіїв $u,v \in \{1, 2, 3, 4\}$, вона є тотожною системі (2.2) або (2.3). Для прикладу, при $u=2$ та $r=4$, система (2.6) набуває вигляду:

$$\begin{cases} a_{2j} + a_{4j} = z_{1j} = s_{1j} + a_{1j} + a_{3j} \\ a_{2j} = z_{2j} = s_{2j} + a_{1j} \end{cases}, \forall j = 1,\dots,m \tag{2.7}$$

Для того, щоб відновити втрачені дані із носіїв 2 та 4, потрібно розв’язати систему рівнянь (2.7), що є тривіальною та обчислювально простою задачею.

Крім того, при втраті доступу до одного загального та одного додаткового носія, втрачені дані також легко відновлюються. Таким чином, у разі втрати доступу до u-того носія, $u \in \{1,\dots,n\}$, та одного із додаткових носіїв, крім першого, його дані відновлюються легко із використанням інформації із першого додаткового носія, у якому зберігається сума за модулем 2 значень блоків усіх інших основних носіїв, дані яких за умовою є непошкодженими. У такому випадку відновлення даних зводиться до простої суми за модулем 2 всіх непошкоджених основних носіїв та першого додаткового носія. З іншого боку, при втраті даних із одного основного u-того носія, $u \in \{1,\dots,n\}$, та першого додаткового носія, у якому зберігається результат обчислення суми за формулою (2.1), на одному із непошкоджених додаткових носіїв обов’язково є збережені дані із u-того носія у вигляді контрольної суми. Тоді для відновлення втрачених даних необхідною є знову ж таки, одна сума за модулем 2 відповідних блоків відповідних носіїв. Для прикладу, якщо втраченим основним носієм є 3-й, тобто $u=3$, а контрольні суми були обчислені за системою (2.6), то втрачені дані із 3-го носія відновлюються використанням контрольних блоків із 3-го додаткового носія за формулою $a_{3j}=s_{3j}+a_{1j}$, $\forall j \in \{1,\dots,m\}$.

Як бачимо, запропонований метод резервування та відновлення даних, що зберігаються на віддалених носіях, надає гарантії відновлення даних при втраті доступу до будь-яких двох носіїв, включаючи як основні, так і додаткові носії, на яких зберігаються контрольні суми. Крім того, обчислення, пов’язані із

резервуванням та відновленням даних є дуже простими та не повинні відзначатися помітно на часі читання та запису інформації на віддалені носії.

Проте, очевидним недоліком цього методу є надто велике значення надмірності резервування даних, що є доволі важливим критерієм алгоритмів резервування даних. Цей недолік можна усунути і для цього пропонується відновлювати одразу декілька пар блоків, а не незалежно відновлювати кожний збережений блок. Справа в тому, що в раніше описаному методі побудови та розв'язання системи із двох булевих рівнянь (2.2), відновлюються значення відповідних блоків із деяких двох носіїв. Проте можливо ускладнити цю систему так, щоб у ній було *2h* булевих рівнянь і її розв'язком було *h* блоків із кожного з двох носіїв, до яких було втрачено доступ.

Наприклад, далі буде розглянуто такий метод для h=2. Нехай доступ було втрачено для носіїв з індексами q та r. Далі буде розглянуто задачу відновлення деякої пари блоків із цих носіїв. Позначимо ці блоки як a_{ql}, a_{ql+1} — для носія q та a_{rl}, a_{rl+1} — для носія r, де $l \in \{1,3,...,m-1\}$. За аналогією до систем (2.2) та (2.3), відновлення втрачених даних із двох носіїв, до яких втрачено доступ, може бути виконане шляхом розв'язання для кожного можливого значення l однієї із чотирьох запропонованих систем булевих рівнянь. При чому для всіх систем першим і другим рівнянням є суми по модулю 2 всіх l-тих блоків загальних носіїв — для першого рівняння та всіх (l+1)-их блоків загальних носіїв — для другого рівняння.

$$\begin{cases} a_{ql} + a_{rl} = g_{1l} \\ a_{q,l+1} + a_{r,l+1} = g_{2l+1} \\ a_{ql} = g_{3l} \\ a_{rl} + a_{r,l+1} = g_{4l} \end{cases}, \forall l \in \{1,3,...,m-1\} \qquad \begin{cases} a_{ql} + a_{rl} = g_{1l} \\ a_{q,l+1} + a_{r,l+1} = g_{2l+1} \\ a_{ql} + a_{q,l+1} = g_{3l} \\ a_{rl} = g_{4l} \end{cases}, \forall l \in \{1,3,...,m-1\}$$

$$\begin{cases} a_{ql} + a_{rl} = g_{1l} \\ a_{q,l+1} + a_{r,l+1} = g_{2l+1} \\ a_{q,l+1} = g_{3l} \\ a_{rl} + a_{r,l+1} = g_{4l} \end{cases}, \forall l \in \{1,3,...,m-1\} \qquad \begin{cases} a_{ql} + a_{rl} = g_{1l} \\ a_{q,l+1} + a_{r,l+1} = g_{2l+1} \\ a_{ql} + a_{q,l+1} = g_{3l} \\ a_{r,l+1} = g_{4l} \end{cases}, \forall l \in \{1,3,...,m-1\} \quad (2.8)$$

де g_{kj} — деяка комбінація контрольної суми j-того блока, що зберігається на k-тому носії та блоків всіх інших непошкоджених основних носіїв.

Для того, аби отримати одну із зазначених систем для кожних специфічних значень q та r потрібно виконати наступні дії.

Як вже зазначалося, перше та друге рівняння із усіх систем (2.8) відповідають рівнянню (2.1), тому для них можна використовувати дані із додаткового носія, у якому якраз і зберігається результат обчислення контрольної суми за рівнянням (2.1).

При цьому отримати решту рівнянь із цих систем таким простим способом не можна, адже на момент резервування даних значення індексів дисків, які відмовлять у майбутньому, q та r є невідомими. З іншого боку, очевидним є той факт, що це решта рівнянь можуть бути отримані із системи булевих рівнянь, що при будь-яких можливих значеннях q та r містить деяке рівняння, в яке входять тільки блоки $a_{ql}+a_{ql+1}$ або a_{rl}.

Для прикладу, розглянемо метод відновлення 2-ох пар блоків для системи, що складається із чотирьох загальних носіїв, тобто для випадку, коли n=4. При цьому, при втраті доступу до двох будь-яких носіїв, для відновлення втрачених даних, необхідним є використання чотирьох додаткових носіїв, у яких зберігаються контрольні суми, що формуються наступним чином:

$$\begin{cases} s_{1,l} = a_{1l} + a_{2l} + a_{3l} + a_{4l} \\ s_{2,l+1} = a_{1,l+1} + a_{2,l+1} + a_{3,l+1} + a_{4,l+1} \\ s_{3l} = a_{1l} + a_{2l} + a_{2,l+1} \\ s_{4l} = a_{1l} + a_{l+1} + a_{3j} \\ s_{5l} = a_{4l} + a_{4,l+1} \end{cases} , \forall l = 1,3,\ldots,m-1 \qquad (2.9)$$

де s_{kj} — контрольна сума j-того блока, що зберігається на k-тому додатковому носії.

Очевидним є те, що дана система є лінійно незалежною, і тому при будь-яких індексах пошкоджених носіїв $r,q \in \{1, 2, 3, 4\}$, вона є тотожною одній

із систем (2.8). Для прикладу, при значеннях індексів пошкоджених носіїв r=4 та q=1, то система (2.9) набуває вигляду:

$$\begin{cases} a_{1l} + a_{4l} = g_{1l} = s_{1l} + a_{2l} + a_{3l} \\ a_{1,l+1} + a_{2,l+1} = g_{2l} + a_{2,l+1} + a_{3,l+1} \\ a_{1l} = g_{3l} = s_{3l} + a_{2l} + a_{2,l+1} \\ a_{4l} + a_{4,l+1} = g_{4l} = s_{5,l} \end{cases}, \forall l = 1,3,\ldots,m-1 \quad (2.10)$$

Очевидним є те, що ця система є тотожною першій системі із рівняння (2.8), а також те, що її розв’язок надає можливість однозначно відновити значення втрачених пар блоків для всіх значень l.

Із системи (2.9) видно, що для того, аби визначити перше та друге рівняння систем, використовується інформація із додаткового носія, блоки в якому обчислені за формулою (2.1). Якщо вважати, що цю роль виконує перший додатковий носій, то у ньому будуть зберігатися m блоків, тоді як на решті додаткових носіїв будуть зберігатися лише половина блоків, тобто $m/2$.

Отже, звідси видно, що при відновленні пар блоків, тобто при $h=2$ збільшується обчислювальна складність алгоритму відновлення даних. Це пов’язано з тим, що замість двох незалежних систем лінійних булевих рівнянь, розв’язується одна, яка, проте, складається із чотирьох рівнянь. При цьому кількість додаткових носіїв не змінюється і обчислюється за формулою (2.5), але виграш у кількості використаної пам’яті досягається за рахунок того, що тільки на першому додатковому носії зберігаються всі m блоків, тоді як на решті зберігається лише їх половина. Такий метод дозволяє зменшити надлишковість операції резервування даних.

2.2. Опис модифікації методу резервування та відновлення даних з двох носіїв з використанням діагональних сум

Розглянемо ту ж модель збереження інформаційних блоків. Пропонується дані, що записуються у вигляді файлу, розділити на n блоків, кожен із яких

записується на відповідний віддалений носій, кількість яких також дорівнює n. Таким чином i-тий блок буде збережено на i-тому носії, де $i \in \{1,...,n\}$. Відповідно, на кожному такому i-тому носії буде всього збережено m блоків. Позначимо одне j-тий блок з i-того носія як a_{ij}, де $i \in \{1,...,n\}$; $j \in \{0,...,m-1\}$. Варто зауважити, що на практиці практично завжди для значень m та n виконується співвідношення $m >> n$.

Пропонується покращений метод резервування даних, який дозволяє відновити дані при їх втраті на будь-яких двох носіях із системи. Варто зауважити, що з теоретичної точки зору, для того, аби мати можливість відновити дані при їх втраті із k віддалених носіїв, дані, які зберігаються на кожному з n дисків повинні бути збережені на k або більше додаткових носіях. Це означає, що для відновлення даних із двох втрачених віддалених носіїв, необхідним мінімумом надлишковості є використання двох додаткових віддалених носіїв.

Запропонований метод передбачає зберігання сум за модулем 2 всіх відповідних блоків відповідних носіїв, що представляють із себе контрольні суми, обчислені за формулою (2.1), на першому додатковому носієві:

$$d_i = a_{1,i} \oplus a_{2,i} \oplus a_{3,i} \oplus ... \oplus a_{n,i}, \forall i = 0,...,m-1. \tag{2.11}$$

Тоді, на другому додатковому носієві, буде зберігатися $m+1$ сума за модулем 2 стовпців наступної матриці, що представляє із себе скошену матрицю всіх блоків всіх носіїв із нульовою головною діагоналлю:

$$\begin{matrix}
0 & a_{1,0} & a_{1,1} & a_{1,2} & \dots & a_{1,n-2} & a_{1,n-1} & \dots & a_{1,m-1} \\
a_{2,m-1} & 0 & a_{2,0} & a_{2,2} & \dots & a_{2,n-3} & a_{2,n-2} & \dots & a_{2,m-2} \\
a_{3,m-2} & a_{3,m-1} & 0 & a_{3,0} & \dots & a_{3,n-4} & a_{3,n-3} & \dots & a_{3,m-3} \\
a_{4,m-3} & a_{4,m-2} & a_{4,m-1} & 0 & \dots & a_{4,n-5} & a_{4,n-4} & \dots & a_{4,m-4} \\
a_{n,m-n+1} & a_{n,m-n+2} & a_{n,m-n+4} & a_{n,m-n-5} & \dots & 0 & a_{n,0} & \dots & a_{n,m-n}
\end{matrix} \tag{2.12}$$

Позначимо ці суми за модулем 2 як q_k, $k \in \{0,...,m\}$. Обчислюватися вони будуть за формулами:

$$\begin{aligned} q_0 &= a_{2,m-1} \oplus a_{3,m-2} \oplus a_{4,m-3} \oplus ... \oplus a_{n,m-n+1} \\ q_1 &= a_{10} \oplus a_{3,m-1} \oplus a_{4,m-2} \oplus ... \oplus a_{n,m-n+2} \\ &..... \\ q_n &= a_{1,n-1} \oplus a_{2,n-2} \oplus a_{3,n-3} \oplus ... \oplus a_{n-1,0} \\ q_{n+1} &= a_{1,n} \oplus a_{2,n-1} \oplus a_{3,n-2} \oplus ... \oplus a_{n,0} \\ &..... \\ q_m &= a_{1,m} \oplus a_{2,m-1} \oplus a_{3,m-2} \oplus ... \oplus a_{n,m-n} \end{aligned} \quad (2.13)$$

Тоді, при відсутності доступу до будь-яких двох носіїв можливі 3 ситуації:

1. Доступ було втрачено до двох основних носіїв із індексами v та w, $v,w \in \{1,...,n\}$.
2. Доступ було втрачено до одного основного носія з індексом v та першого додаткового носія.
3. Доступ було втрачено до одного основного носія з індексом v та другого додаткового носія.

Розглянемо всі ці ситуації по черзі. При втраті доступу до двох основних носіїв із індексами v та w, відновити відповідно потрібно блоки, що знаходяться саме на цих носіях. Позначимо ці блоки як a_{vj} та a_{wj}, де $j=0,...,m-1$. Для відновлення цих блоків можуть бути використані контрольні суми, які зберігаються на додаткових носіях, а також блоки усіх непошкоджених дисків. Таким чином, із першого додаткового носія можна отримати m рівнянь для кожного відповідного блока, що матимуть у якості невідомих лише блоки двох втрачених дисків, і матимуть наступний вигляд:

$$a_{v,i} \oplus a_{w,i} = \xi_i = d_i \oplus \sigma_{i,v,w}, \forall i = 0,..., m-1 \quad (2.14)$$

де $\sigma_{i,v,w}$ – сума за модулем 2 всіх i-тих блоків всіх непошкоджених основних носіїв.

Схожим чином, матрицю (2.12) можна переписати так, аби вона містила лише 2 рядки із невідомими даними. У поточному випадку це будуть рядки з

номерами v та w, які містять у собі відповідні втрачені блоки a_{vj} та a_{wj}, де $j=0,...,m-1$:

$$\begin{matrix} a_{v,m-v+1} \dots a_{v,m-1} & 0 & a_{v,0} & \dots & a_{v,g-1} & a_{v,g} & a_{v,g+1} \dots a_{v,m-v} \\ a_{w,m-w+1} \dots a_{w,m-g-1} & a_{w,m-g} & a_{w,m-g+1} & \dots & 0 & a_{w,0} & a_{w,1} \dots a_{w,m-w} \end{matrix} \quad (2.15)$$

де $v<w$ та $g=w-v$.

Суми всіх i-тих стовпців β_i цієї модифікованої матриці представляють із себе суму за модулем 2 блока q_i другого допоміжного носія та $\delta_{i,v,w}$ — суми блоків із i-того стовпця матриці (2.12) всіх непошкоджених носіїв. Таким чином $\beta_i=q_i \oplus \delta_{i,v,w}$. Крім того, у матриці (2.15) є два стовпця, в яких одне із двох блоків дорівнює нулю (конкретно це $v-1$ та $w-1$ стовпці):

$$\begin{aligned} a_{w,m-g} &= \beta_{v-1} = q_{v-1} \oplus \delta_{v-1,m,v} \\ a_{v,g-1} &= \beta_{w-1} = q_{w-1} \oplus \delta_{w-1,m,v} \end{aligned} \quad (2.16)$$

Для інших стовпців матриці (2.15) не виконується ця властивість, тому рівняння, що їм відповідають, включають два невідомих блока:

$$\begin{aligned} &a_{v,u-v} \oplus a_{w,u-w} = \beta_u = q_u \oplus \delta_{u,m,v}, \ \forall u \in \{w,..,m\} \\ &a_{v,m-v+l+1} \oplus a_{w,m-w+l+1} = \beta_l = q_l \oplus \delta_{l,m,v}, \ \forall l \in \{0,..,v-2\} \\ &a_{v,e-m} \oplus a_{w,m-w+e+1} = \beta_e = q_e \oplus \delta_{e,m,v}, \ \forall e \in \{v,..,w-2\} \end{aligned} \quad (2.17)$$

Алгоритм відновлення елементів носіїв, до яких втрачено доступ, a_{vj} та a_{wj}, де $j=0,...,m-1$, є наступним:

1. Через те, що рівняння під номером $(v-1)$, яке має вигляд $a_{w,m-g}=\beta_{v-1}=q_{v-1} \oplus \delta_{v-1,m,v}$, включає в себе лише один невідомий блок $a_{w,m-g}$, це рівняння вибирається як стартове. Крім цього, ведеться індекс поточного рівняння u, який стає рівним $(v-1)$. Також ведеться індекс поточного блока j, який встановлюється в $m-g$. На цьому ж етапі обчислюється значення елемента $a_{w,j}$ із вищевикладеної формули.
2. Обчислюється значення блока $a_{v,j}$ із (2.14): $a_{v,j}=\xi_{v-1} \oplus a_{w,j}$.
3. Знаходиться індекс наступного поточного рівняння, в яке входить обчислений на попередньому кроці блок $a_{v,j}$, за формулою: $u=(m+1+u-g) mod (m+1)$.

4. Знаходиться індекс i блока $a_{w,i}$, який входить в u-те рівняння: $i=(m+1+j-g) mod (m+1)$.
5. Обчислюється значення блока $a_{w,i}=\beta_u \oplus a_{v,j}$. Оновлюється індекс поточного блока: $j=i$.
6. Перевіряється індекс поточного блока — якщо $j=g-1$ — кінець, інакше повернення на етап 2.

Очевидним є те, що поданий алгоритм надає можливість послідовно відновити всі втрачені блоки a_{vj} та a_{wj}, де $j=0,...,m-1$, які зберігаються на носіях з індексами v та w. Надається ця можливість лише за умови, що зміщення індекса поточного блока $m+1-g$ не має спільних дільників із модулем рівняння $m+1$. Через те, що інформація про те, яка пара носіїв буде втрачена, є невідомою наперед, значення зміщення $m+1-g$ може приймати різноманітні значення. Проте для того, аби ця умова виконувалась незалежно від значень v та w, потрібно, аби число $m+1$ було простим, тобто аби воно взагалі не мало дільників [38].

Отже, це доводить те, що при втраті доступу до двох основних віддалених носіїв, описаний метод резервування забезпечує відновлення усіх втрачених даних, які на них зберігались.

Розглянемо варіант, коли доступ втрачно до одного основного носія з індексом v та першого додаткового носія, на якому зберігається контрольна сума, розрахована за формулою (2.1). У такому випадку відновленню підлягає лише основний носій з індексом v. При цьому кожна сума q_i, $i=0,...,m$, яка міститься на другому додатковому носієві, може бути виражена у вигляді:

$$\begin{aligned} q_i &= a_{v,m-v+i-1} \oplus \eta_{i,v}\,, \ \forall i \in \{0,\ldots,v-1\} \\ q_l &= a_{v,l-v} \oplus \eta_{l,v}\,, \ \forall l \in \{v,\ldots,m\}, \end{aligned} \qquad (2.18)$$

де $\eta_{j,v}$ є сумою усіх непошкоджених елементів j-того стовпця матриці (2.12), тобто всіх крім тих, що знаходяться у рядку v. Таким чином блоки носія з індексом v — a_{vj}, $j=0,...,m-1$ — які було втрачено, відновлюються за формулою:

$$a_{v,j} = q_{(j+v) \bmod (m+1)} \oplus \eta_{(j+v) \bmod (m+1),v}\,, \ \forall j \in \{0,\ldots,m-1\} \qquad (2.19)$$

І нарешті, при третьому розвитку подій, коли доступ втрачно до одного основного носія з індексом *v* та другого додаткового носія, який містить суми, обчислені за формулами (2.13), відновити дані необхідно лише з *v*-того носія. При цьому перший додатковий носій також залишається доступним та цілим, а також, суми, які на ньому зберігаються включають в себе блоки всіх основних носіїв. Відповідним чином, блоки носія з індексом *v* — a_{vj}, *j=0,...,m-1* — які було втрачено, відновлюються таким чином:

$$a_{v,j} = d_j \oplus a_{1,j} \oplus \ldots \oplus a_{v\text{-}1,j} \oplus a_{v+1,j} \oplus \ldots \oplus a_{n,j},\ \forall j \in \{0,..,m\text{-}1\} \quad (2.20)$$

Виводиться ця формула із (2.1) та її обчислення є тривіальним.

Отже, це доводить те, що при втраті доступу до будь-яких двох віддалених носіїв, описаний метод резервування забезпечує відновлення усіх втрачених даних, які зберігались на основних носіях.

2.3. Розробка модифікації методу зі спрощеною процедурою відновлення даних із трьох носіїв

Зроблений аналіз показує, що є можливість значного спрощення методу резервування та відновлення даних із трьох носіїв із втраченим доступом до них, за рахунок невеликого зростання кількості контрольних даних на додаткових дискових носіях. Пропонується наступний підхід.

Береться схожа модель зберігання даних та виникнення помилок. Пропонується дані, розділити на *n* блоків, кожен із яких записується на відповідний віддалений носій, кількість яких також дорівнює *n*. Так *i*-тий блок буде збережено на *i*-тому носії, де $i \in \{1,...,n\}$. Відповідно, на кожному такому *i*-тому носії буде всього збережено *m* блоків. Позначимо один *j*-тий блок з *i*-того носія як s_{ij}, де $i \in \{1,...,n\}$; $j \in \{1,...,m\}$. Варто зауважити, що на практиці практично завжди для значень *m* та *n* виконується співвідношення $m >> n$. Вважається, що кількість втрачених носіїв підпорядковується біноміальному закону розподілу. Це говорить про те, що втрати доступу до носіїв є незалежними, а також те, що

ймовірність втрати двох носіїв є на порядок меншою за ймовірність втрати одного, ймовірність втрати трьох — на порядок меншою за ймовірність втрати двох і так далі. Це підводить до того, що для спрощення методу резервування та відновлення втрачених даних, що може дозволити розробку більш ефективного способу, варто обмежити кількість носіїв, що можуть бути одночасно втраченими. Для цього методу максимальна кількість втрачених дисків дорівнює трьом, що практично повністю покриває множину практичних ситуацій, які можуть виникнути.

У запропонованому методі для резервування та відновлення до трьох втрачених носіїв даних будуть використовуватися три додаткових носія. Блоки цих носіїв пропонується будувати наступним чином. Розглянемо наступну матрицю, що містить всі блоки усіх основних носіїв системи:

$$\begin{matrix} S_{1,1} & S_{1,2} & S_{1,3} & S_{1,4} & \dots. & S_{1,m-1} & S_{1,m} \\ S_{2,1} & S_{2,2} & S_{2,3} & S_{2,4} & \dots. & S_{2,m-1} & S_{2,m} \\ S_{3,1} & S_{3,2} & S_{3,3} & S_{3,4} & \dots. & S_{3,m-1} & S_{3,m} \\ & & & \dots\dots & & & \\ S_{n,1} & S_{n,2} & S_{n,3} & S_{n,4} & \dots. & S_{n,m-1} & S_{n,m}, \end{matrix} \quad (2.21)$$

Тоді у додаткових носіях будуть зберігатися суми за модулем два всіх стовпців матриці (2.21) — у першому, суми за модулем два основних діагоналей матриці (2.21) — у другому та суми за модулем два побічних діагоналей матриці (2.21) — у другому.

Таким чином, перший додатковий носій буде містити контрольні суми c_i, $i=1,...,m$. Формуватися ці суми будуть за допомогою обчислення суми за модулем 2 кожного стовпця матриці (2.21), що відповідає сумуванню відповідних блоків усіх основних носіїв, що рахуються за формулою (2.1). Саме для розглянутого випадку:

$$\forall i=1,\dots,m: c_i = s_{1,i} \oplus s_{2,i} \oplus s_{3,i} \oplus \dots \oplus s_{n,i} \quad (2.22)$$

На другому додатковому носієві будуть міститися $m+n$-1 контрольних сум d_i, $i=1,...,m+n-1$. Ці суми будуть формуватися сумуванням за модулем два всіх блоків головних діагоналей матриці (2.21):

$$
\begin{aligned}
d_1 &= s_{n,1} \\
d_2 &= s_{n-1,1} \oplus s_{n,2} \\
d_2 &= s_{n-2,1} \oplus s_{n-1,2} \oplus s_{n,3}\ m \\
&\ldots. \\
d_{n-1} &= s_{2,1} \oplus s_{3,2} \oplus \ldots \oplus s_{n,n-1} \\
d_n &= s_{1,1} \oplus s_{2,2} \oplus \ldots \oplus s_{n,n} \\
d_{n+1} &= s_{1,2} \oplus s_{2,3} \oplus \ldots \oplus s_{n,n+1} \\
&\ldots. \\
d_m &= s_{1,m-n+1} \oplus s_{2,m-n+2} \oplus \ldots \oplus s_{n,m} \\
d_{m+1} &= s_{1,m-n+2} \oplus s_{2,m-n+3} \oplus \ldots \oplus s_{n-1,m} \\
d_{m+2} &= s_{1,m-n+3} \oplus s_{2,m-n+4} \oplus \ldots \oplus s_{n-2,m} \\
&\ldots. \\
d_{m+n-2} &= s_{1,m-1} \oplus s_{2,m} \\
d_{m+n-1} &= s_{1,m}
\end{aligned}
\qquad (2.23)
$$

На третьому додатковому носії, аналогічним чином, будуть зберігатися $m+n$-1 контрольних сум q_i, $i=1,...,m+n-1$. Ці суми будуть формуватися сумуванням за модулем два всіх блоків побічних діагоналей матриці (2.21).

Тоді, коли доступ втрачається до трьох носіїв із n, можливі три ситуації:

1. Втрачено було три основних носія із індексами v, u та w, $v<u<w$, $v,u,w \in \{1,...,n\}$.
2. Втрачено було два основних носія з індексами v та w, $v<w$, $v,w \in \{1,...,n\}$ та один із додаткових носіїв.
3. Втрачено було один основний носій з індексом v та два додаткові носії.

Розглянемо ці ситуації по черзі.

Тоді, коли втраченими є три основних носія з індексами v, u та w, відновленню підлягають блоки s_{vj}, $j=1,...,m$, які зберігаються на v-тому носієві,

блоки s_{uj}, $j=1,...,m$, які зберігаються на u-тому носієві та блоки s_{wj}, $j=1,...,m$, які зберігаються на w-тому носієві. При цьому контрольні суми, що збережені на першому додатковому носієві можуть бути виражені у вигляді:

$$\forall j=1,\ldots,m: c_j = s_{v,j} \oplus s_{u,j} \oplus s_{w,j} \oplus \delta_j \quad (2.24)$$

де δ_j є сумою за модулем два всіх відповідних блоків з індексом j всіх непошкоджених носіїв даних:

$$\delta_j = \bigoplus_{i=1, j\neq v,u,w}^{n} s_{i,j} \quad (2.25)$$

Враховуючи це, контрольні суми із другого додаткового носія можуть бути виражені наступним чином:

$$\forall j=1,\ldots,m: d_{n-w+j} = s_{w,j} \oplus \eta_{w,j} \oplus \xi_{w,j} \quad (2.26)$$

де $\eta_{w,j}$ є сумою всіх блоків із головної діагоналі матриці (2.21), яка включає в себе блок $s_{w,j}$, які розташовані вище рядка матриці (2.21) з індексом w,

$\xi_{w,j}$ є сумою всіх блоків із головної діагоналі матриці (2.21), яка включає в себе блок $s_{w,j}$, які розташовані нижче рядка матриці (2.21) з індексом w.

Відповідно обчислюється значення $\eta_{w,j}$ наступним чином:

$$\begin{aligned}
&\eta_{w,j} = 0, \text{ якщо } j = 1 \\
&\eta_{w,j} = s_{w-1,j-1} \oplus s_{w-2,j-2} \oplus \ldots \oplus s_{w-j+1,1} \text{ якщо } 1 < j < w \\
&\eta_{w,j} = s_{w-1,j-1} \oplus s_{w-2,j-2} \oplus \ldots \oplus s_{1,j-w+1}, \text{ якщо } j \geq w,
\end{aligned} \quad (2.27)$$

а значення $\xi_{w,j}$ обчислюється як:

$$\begin{aligned}
&\xi_{w,j} = 0, \text{ якщо } j = m \\
&\xi_{w,j} = s_{w+1,j+1} \oplus s_{w+2,j+2} \oplus \ldots \oplus s_{n,\, n-w+j}, \text{ якщо } j \leq m+n-w \\
&\xi_{w,j} = s_{w-1,j-1} \oplus s_{w-2,j-2} \oplus \ldots \oplus s_{w,m}, \text{ якщо } j > m+n-w
\end{aligned} \quad (2.28)$$

Таким же чином, контрольні суми із третього додаткового носія можуть бути виражені наступним чином:

$$\forall j=1,\ldots,m: q_{v+j-1} = s_{v,j} \oplus \gamma_{v,j} \oplus \lambda_{v,j} \quad (2.29)$$

де $\gamma_{v,j}$ є сумою всіх блоків із побічної діагоналі матриці (2.21), яка включає в себе блок $s_{v,j}$, які розташовані нижче рядка матриці (2.21) з індексом v;

$\lambda_{v,j}$ є сумою всіх блоків із побічної діагоналі матриці (2.21), яка включає в себе блок $s_{v,j}$, які розташовані вище рядка матриці (2.21) з індексом v.

Відповідно обчислюється значення $\gamma_{v,j}$ наступним чином:

$$\begin{aligned}
&\gamma_{v,j} = 0, \text{ якщо } j = 1 \\
&\gamma_{v,j} = s_{v+j-1,1} \oplus s_{v+j-1,2} \oplus \ldots \oplus s_{v+1,j-1}, \text{ якщо } 1 < j < n\text{-}v+1 \\
&\gamma_{w,j} = s_{n,j+v-n} \oplus s_{n-1,j+v-n-1} \oplus \ldots \oplus s_{v+1,j-1}, \text{ якщо } j \geq n\text{-}v+1
\end{aligned} \qquad (2.30)$$

а значення $\lambda_{v,j}$ обчислюється як:

$$\begin{aligned}
&\lambda_{v,j} = 0, \text{ якщо } j = m \\
&\lambda_{v,j} = s_{v-1,j+1} \oplus s_{v-2,j+2} \oplus \ldots \oplus s_{1,j+v-1}, \text{ якщо } j \leq m\text{-}v+1 \\
&\lambda_{v,j} = s_{v-1,j+1} \oplus s_{v-2,j+2} \oplus \ldots \oplus s_{j+v-m,m}, \text{ якщо } j > m\text{-}v+1
\end{aligned} \qquad (2.31)$$

Із рівняння (2.29) слідує, що при j=1 сума q_v із третього додаткового носія може бути виражена у вигляді: $q_v = s_{v,1} \oplus \lambda_{v,1}$. При цьому контрольна сума q_v на момент відновлення відома, як і $\lambda_{v,1}$, так як за визначенням індекс носія v є найменшим із усіх індексів втрачених носіїв. Відповідно всі блоки із носіїв з індексами, меншими за v є непошкодженими, і тому значення блоків s_{ij}, $i=1,...,v\text{-}1;\ j=2,...,v$ є відомими. Відповідно перший блок $s_{v,1}$ втраченого віддаленого носія v відновлюється із формули:

$$s_{v,1} = q_v \oplus \lambda_{v,1} = q_v \oplus s_{v-1,2} \oplus s_{v-2,3} \oplus \ldots \oplus s_{1,v}. \qquad (2.32)$$

Схожим чином, із рівняння (2.26) слідує, що тоді, коли j=1, сума d_{n-w+1} із другого додаткового носія , може бути виражена як: $d_{n-w+1} = s_{w,1} \oplus \xi_{w,1}$. При цьому контрольна сума d_{n-w+1} на момент відновлення відома, як і $\xi_{w,1}$, так як за визначенням індекс носія w є найбільшим із усіх індексів втрачених носіїв. Відповідно всі блоки із носіїв з індексами, більшими за w є непошкодженими, і тому значення блоків s_{ij}, $i=w+1,...,n;\ j=2,...,n\text{-}w+1$ є відомими. Відповідно перший блок $s_{w,1}$ втраченого віддаленого носія w відновлюється із формули:

$$s_{w,1} = d_{n-w+1} \oplus \xi_{v,1} = d_{n-w+1} \oplus s_{w+1,2} \oplus s_{w+2,3} \oplus \ldots \oplus + s_{n,n-w+1} \qquad (2.33)$$

Аналогічно, перший блок $s_{u,1}$ втраченого третього носія з індексом u, буде відновлюватись із використанням контрольних сум, збережених на першому

носієві, та обчислених за формулою (2.25). Відповідна формула для обчислення першого блока носія *u* є наступною:

$$s_{u,1} = c_1 \oplus s_{v,1} \oplus s_{w,1} \oplus \delta_1 \quad (2.34)$$

Видно, що запропонований алгоритм вже дозволяє відновити перші блоки втрачених носіїв з індексами *v, u, w.*

Відновлення решти блоків із цих носіїв відбувається послідовно. Таким чином наступними відновлюються блоки $s_{v,2}$, $s_{w,2}$, $s_{u,2}$, для яких уже все доступно, адже решта невідомих блоків були обчислені на попередньому етапі. Тоді аналогічно відновлюються треті блоки $s_{v,3}$, $s_{w,3}$, $s_{u,3}$ і далі, аж до останніх блоків під номерами *m* — $s_{v,m}$, $s_{w,m}$, $s_{u,m}$. Загальний алгоритм відновлення блоків під індексами *j* із втрачених віддалених носіїв із номерами *v*, *w* та *u*, які можна позначити як s_{vj}, s_{wj}, s_{uj}, описано далі. При цьому варто зазначити, що вважається, що $v<u<w$.

1. Блок $s_{v,j}$ із втраченого основного віддаленого носія під номером *v* відновлюється з допомогою формули (2.29):

$$s_{v,j} = q_{v+j-1} \oplus \gamma_{v,j} \oplus \lambda_{v,j} \quad (2.35)$$

   У цій формулі значення q_{v+j-1} є відомим, так як це попередньо обчислена контрольна сума, збережена на третьому додатковому носієві. Також через те, що всі блоки із суми $\lambda_{v,j}$, які розташовані на побічній діагоналі матриці (2.21), що містить $s_{v,j}$, належать до носіїв із індексами, меншими за *v,* не є втраченими за умовою, а отже, відомими, а також всі блоки із суми $\gamma_{v,j}$, які розташовані на побічній діагоналі матриці (2.21), що містить s_{vj}, є блоками з індексами, меншими за *j*, а отже відомими через виконання попередніх етапів обчислень, єдиним невідомим у рівнянні (2.35) залишається блок $s_{v,j}$, який легко знайти, виконавши відповідне підсумування. Так відновлюються блоки носія *v.*

2. Блок $s_{w,j}$ із втраченого основного віддаленого носія під номером *w* відновлюється з допомогою формули (2.26):

$$s_{w,j} = d_{n-w+j} \oplus \eta_{w,j} \oplus \xi_{w,j}, \quad (2.36)$$

У цій формулі значення d_{n-w+j} є відомим, так як це попередньо обчислена контрольна сума, збережена на другому додатковому носієві. Також через те, що всі блоки із суми $\xi_{w,j}$, які розташовані на головній діагоналі матриці (2.21), що містить s_{wj}, належать до носіїв із індексами, більшими за w, не є втраченими за умовою, а отже, відомими, а також всі блоки із суми $\eta_{w,j}$, які розташовані на головній діагоналі матриці (2.21), що містить s_{wj}, є блоками з індексами, меншими за j, а отже відомими через виконання попередніх етапів обчислень, єдиним невідомим у рівнянні (2.36) залишається блок $s_{w,j}$, який легко знайти, виконавши відповідне підсумування. Так відновлюються блоки носія w.

3. Блок $s_{u,j}$ із втраченого основного віддаленого носія під номером u відновлюється з допомогою формули (2.34):

$$s_{u,j} = c_j \oplus s_{v,j} \oplus s_{w,j} \oplus \delta_j \qquad (2.37)$$

У цій формулі значення c_j є відомим, так як це попередньо обчислена контрольна сума, збережена на першому додатковому носієві. Також через те, що всі блоки із суми δ_j, належать до непошкоджених носіїв, а отже, є відомими, а також блоки $s_{w,j}$ та $s_{v,j}$ є відомими через виконання попередніх етапів обчислень, єдиним невідомим у рівнянні (2.37) залишається блок $s_{u,j}$, який легко знайти, виконавши відповідне підсумування. Так відновлюються блоки носія u.

При іншій ситуації, коли доступ було втрачено до двох основних віддалених носіїв з індексами v, w; $v<w$, та одного додаткового носія, методи відновлення даних залежать від конкретного додаткового носія, до якого було втрачено доступ:

1. Якщо втрачено було перший додатковий віддалений носій, на якому збережені суми, пораховані за формулою (2.1), то відновлення блоків s_{vj} носія v та s_{wj} носія w, де $j \in \{1,...,m\}$, відбувається за алгоритмом, викладеним вище, проте без виконання пункту 3.
2. Якщо втрачено було другий додатковий віддалений носій (на якому зберігаються суми головних діагоналей матриці (2.21)), то відновлення

блоків s_{vj} носія v, де $j \in \{1,...,m\}$ відбувається відповідно до пункту 1 викладеного вище алгоритму, а відновлення блоків s_{wj} носія w, де $j \in \{1,...,m\}$ відбувається відповідно до пункту 3 викладеного вище алгоритму.

3. Якщо втрачено було третій додатковий віддалений носій (на якому зберігаються суми побічних діагоналей матриці (2.21)), то відновлення блоків s_{wj} носія w, де $j \in \{1,...,m\}$ відбувається відповідно до пункту 2 викладеного вище алгоритму, а відновлення блоків s_{vj} носія v, де $j \in \{1,...,m\}$ відбувається відповідно до пункту 3 викладеного вище алгоритму.

Нарешті, при ситуації, коли доступ було втрачено до одного основного віддаленого носія з індексом v та двох додаткових носіїв, методи відновлення даних також залежать від конкретних додаткових носіїв, до яких було втрачено доступ:

1. Якщо додатковим носієм, до якого доступ втрачено не було, є перший додатковий носій, на якому збережені суми, пораховані за формулою (2.1), то відновлення всіх блоків носія v відбувається із використанням сум першого додаткового носія, та для цього використовується формула (2.37).
2. Якщо додатковим носієм, до якого доступ втрачено не було, є другий додатковий носій, на якому зберігаються суми головних діагоналей матриці (2.21), відновлення втрачених даних відбувається з використанням сум другого додаткового носія та формули (2.36).
3. Якщо додатковим носієм, до якого доступ втрачено не було, є третій додатковий носій, на якому зберігаються суми побічних діагоналей матриці (2.21), відновлення втрачених даних відбувається з використанням сум третього додаткового носія та формули (2.35).

Отже, розроблений метод дозволяє одночасно відновлювати три будь-яких втрачених віддалених носія із використанням трьох додаткових віддалених носіїв.

2.4 Можливості реалізації та аналіз ефективності методів відновлення та резервування даних

Як вже згадувалося, на практиці доволі точною моделлю відмови носіїв є те, що кількість відмовлених носіїв за певний час є числом випадковим, яке підпорядковується біноміальному закону розподілу. На практиці кількість відмовлених носіїв за короткий період часу реально не є більшим за 1-2 та практично ніколи не є більшим за 3.

Із викладеного вище опису методу резервування та відновлення даних із віддалених носіїв випливає, що під час відновлення кожного блока із m для кожного втраченого носія обчислюються лише суми за модулем два. Ці операції відповідають рівнянням (2.35-2.37). Через те, що кількість членів у цих рівняннях не перевищує n, то очевидним є те, що час відновлення всіх блоків одного носія буде дорівнювати $T_1 = n \cdot m \cdot t_{XOR}$, де t_{XOR} часом виконання однієї операції додавання за модулем два. Відповідно, час відновлення всіх блоків усіх носіїв обчислюється з рівняння:

$$T = 3 \cdot n \cdot m \cdot t_{XOR} \tag{2.38}$$

Таким чином часова складність виконання операції відновлення даних дорівнює $O(nm)$, де n — кількість основних носіїв, а m — кількість блоків даних на одному носієві. Звідси видно, що вигідно збільшувати розмір одного блоку для зменшення m та зменшення часу відновлення даних, проте надто велике значення розміру одного блоку викликає труднощі, так як при записі розраховується, що один запис даних на диск повинен містити у собі цілу кількість блоків. У такому випадку для такої великих значень розмірів блоків потрібно додатково встановлювати буфери або доповнювати неповний блок нулями чи іншими послідовностями, що будуть говорити про те, що це розширення основних даних.

Іншим важливим критерієм алгоритмів резервування та відновлення даних є надлишковість даних. Для запропонованого методу, загальний об’єм даних, що зберігаються на основних носіях, складає $M_1 = m \cdot n \cdot h$, де h —

довжина одного блоку даних. Запропонований метод використовує три додаткові носії, із яких два — другий та третій — містять $m+n$-1 блоків, що дорівнює кількості діагоналей в матриці (2.21), а перший зберігає у собі m блоків. Звідси слідує, що об’єм надлишкової контрольної інформації складає:

$$M_c = (2 \cdot (m + n - 1) + m) \cdot h = (3 \cdot m + 2 \cdot (n - 1)) \cdot h, \qquad (2.39)$$

де h — розмір одного блоку даних, n — кількість основних носіїв, а m — кількість блоків даних на одному носієві.

Одним із найпопулярніших алгоритмів резервування та відновлення даних на сьогодні є коди Ріда-Соломона [39], тому запропонований метод буде порівняно саме з ними. Нехай l — це довжина в бітах одного блока даних. Тоді максимальний розмір носія за умови використання кодів Ріда-Соломона буде становити $p = 2^l - 1$ блоків. Варто зауважити, що цей розмір включає в себе контрольні блоки, тому реальний максимальний розмір диска буде дещо меншим. Якщо вважати, що використаний код Ріда-Соломона дозволяє відновити k втрачених носіїв, то реальний максимальний розмір основних носіїв буде складати $p_i = p - k \cdot l \cdot m$. Час T_S обчислення k синдромів для кодів Ріда-Соломона обчислюється як $T_S = k \cdot p \cdot (t_M + t_{XOR})$, де t_M — час виконання однієї операції множення на полях Галуа, а t_{XOR} — час виконання однієї операції додавання на полях Галуа, що є операцією додавання за модулем два. Якщо враховувати те, що $t_m \approx 2 \cdot l \cdot t_{XOR}$ [40], то час обчислення синдрому можна виразити як $T_S = 3 \cdot k \cdot p \cdot l \cdot t_{XOR}$. Варто зауважити, що час виконання корекції є значно більшим за час обчислення синдромів. Проте, у випадку відновлення даних із носіїв, часом корекції можна знехтувати. Тоді повний час обробки даних буде дорівнювати $T_R = \frac{m \cdot n \cdot l}{l \cdot (2^l - 1 - k)} \cdot T_S = \frac{6 \cdot m \cdot n \cdot l \cdot p \cdot t_{XOR}}{p - k}$. Асимптотична обчислювальна складність при відновленні кодами Ріда-Соломона є такою ж як і в запропонованому методі *O(mn)* [41], проте зі співвідношення $\frac{T_R}{T} = 2 \cdot l \cdot \frac{p}{p-k}$ видно, що запропонований метод є ефективнішим при збільшенні кількості пошкоджених блоків на носіях.

Крім того, вартим уваги є те, що запропонований метод резервування та відновлення даних, що зберігаються на віддалених носіях є значно простішим в порівнянні з методами кодів Ріда-Соломона і з точки зору логіки, і з точки зору операцій, що виконуються.

Висновки до розділу 2

У результаті виконання досліджень, спрямованих на розробку методів резервування та відновлення двох та трьох носіїв можна зробити наступні висновки:

1. Теоретично обґрунтовано метод резервування та відновлення даних із двох втрачених носіїв із використанням лінійних кодів. Запропоновано алгоритми резервування та відновлення даних для обґрунтованого методу. Показано приклад роботи методу на системі із чотирьох дисків, що дозволив відновити дані із будь-яких двох втрачених. Також показано те, що запропонований метод не є надто ефективним через значно збільшену надлишковість даних, що складає $2+log_2 n$, де n — кількість основних носіїв. Запропоновано покращення цього методу обчисленням контрольних сум для пар блоків, що, хоча і зменшує надлишковість системи майже у два рази, значно ускладнює обчислення резервування та відновлення даних через ускладнення відповідної системи лінійних рівнянь.

2. Запропоновано модифікацію описаного методу відновлення даних із двох втрачених дисків, використовуючи діагональні суми. Теоретично обґрунтовано такий підхід до відновлення даних із двох втрачених носіїв. Доведено те, що цей спосіб завжди дозволяє відновити дані із будь-яких двох втрачених носіїв. Показано, що надлишковість даних при використанні цього методу є близькою до мінімальної та становить 2 носії + 1 блок, а обчислення є тривіальними із часовою складністю $O(nm)$. Запропоновано повний алгоритм відновлення всіх блоків із двох втрачених основних носіїв та одного втраченого й одного додаткового носія. Показано, що в цього методу є обмеження, що накладаються на кількість блоків даних на одному носієві m та передбачають, що $m+1$ повинне бути простим числом для правильної його роботи.

3. На основі запропонованого методу розроблено спрощений спосіб відновлення втрачених даних із трьох носіїв із використанням діагональних сум. Теоретично обґрунтовано використання сум головних та побічних

діагоналей матриці даних для відновлення будь-яких трьох втрачених носіїв із системи. Показано, що розроблений метод однозначно дозволяє відновити дані із будь-яких трьох втрачених носів. Запропоновано повний алгоритм послідовного відновлення всіх блоків всіх основних втрачених носіїв. Запропоновано алгоритми, коли втраченими є один або два додаткових носія. Показано, що даний алгоритм має надлишковість на $2(n-1)$ блоків більшу, ніж теоретично мінімальна, проте використовує прості обчислення, що дозволяє значно прискорити процедури резервування та відновлення.

4. Проведено порівняння розробленого алгоритму відновлення трьох втрачених носіїв із кодами Ріда-Соломона. Показано, що хоча розроблений метод має незначно більшу надлишковість даних та теоретично ту ж асимптотичну складність $O(mn)$, він, проте, є значно швидшим та має меншу часову складність у порівнянні з кодами Ріда-Соломона через використання простих операцій.

РОЗДІЛ 3
РОЗРОБКА ПРОГРАМНОЇ РЕАЛІЗАЦІЇ МЕТОДУ РЕЗЕРВУВАННЯ ТА ВІДНОВЛЕННЯ ПОМИЛОК ПРИ ЇХ ЗБЕРІГАННІ НА ВІДДАЛЕНИХ НОСІЯХ

3.1. Вибір та обґрунтування мови та оточення програмування

Для розробки програмного продукту, що реалізує розроблений метод було використано мову Go та оточення програмування VS Code, що має одну із кращих підтримок цієї мови. Саме мову програмування Go було обрано через декілька причин, а саме:

- Низькорівневість — Go не накладає велику кількість абстракцій для програміста, зокрема дає можливість працювати із покажчиками.
- Швидкість виконання програм — частково через низькорівневість, частково через вбудовану невимушену підтримку паралельності, Go є зараз однією із найбільш ефективних в плані швидкості мов програмування.
- Безпечність — хоча ця мова і надає доступ до низькорівневих засобів програмування, небезпечні конструкції в коді розпізнаються та дають можливість програмісту писати більш надійний код. Ця властивість дає Go значну перевагу перед іншими низькорівневими мовами, такими як C та *C++*.
- Низькі вимоги — Go не потребує значних ресурсів системи та програми, написані на ній можна скомпілювати під фактично будь-яку актуальну платформу.

Одним із недоліків Go є слабка підтримка графічних інтерфейсів, тому розроблена програма використовує термінальний інтерфейс, який щонайменше не програє графічному у користувацькому досвіді, а для деяких випадків є і більш зручним.

Єдиною вимогою для використання написаної програми є наявність Go версії 1.14 чи вище. При цьому необхідним це є лише для компілювання програми. Для виконання бінарного файла не потрібно встановлювати нічого, тому за потреби програма може бути скомпільована та роздана окремо.

3.2. Опис структури програмного продукту

Структура розробленого продукту включає декілька модулів-пакетів. Така структура зручно ділить код на різні зони відповідальності, що спрощує читання та внесення правок в код. Загальна структура програми передбачає виконання різних команд через командний рядок (детально ці команди розглянуті у підрозділі 3.3). При цьому програма моделює роботу із віддаленими носіями даних, та для цієї моделі використовує хотову файлову систему, на якій створює файли, що відповідають дискам, та оперує з ними, як із віддаленими носіями даних.

1. Пакет *main* містить один файл — *main.go*, який є точкою входу програми. Така структура є примусовою для *Go*, адже лише файли із пакета *main* та з функцією *func main()* можуть компілюватися у виконувані бінарні файли. У цьому файлі обробляються аргументи командного рядка з допомогою стандартної бібліотеки *flag*. В залежності від цих аргументів викликаються функції для виконання відповідної дії. У цьому ж модулі, для зручності користувача, при неправильно введених аргументах програма пише у *stderr* помилку та правильне використання програми. Також тут же відбувається обробка помилок та їх виведення, що могли виникнути при виконанні будь-якої дії. Цей модуль імпортує конкретні пакети, що реалізують можливі дії та стан програми.
2. Пакет *diskgroup* містить 3 файли — *disk.go*, *diskgroup.go* та *offsets.go*, які виначають структури та функції для моделювання доступу до носіїв даних. Таким чином, для групи дисків у цьому пакеті визначена структура *DiskGroup*, що містить назву групи *Name* (для зручнішого оперування

нею), кількість дисків у групі, що є її розміром *Size*, слайс (структура даних у *Go*, дещо схожа із масивами із інших мов програмування) із вказівниками на структури дисків *Disks* та структура із даними про збережені файли *FileOffsets*. Крім цього, для цих структур визначені декілька функцій та методів для роботи з ними. Це включає:

2.1. Функція *NewGroup*, що відповідає за створення нової групи дисків із заданою назвою та величиною із мінімальною алокацією пам’яті для даних. Це можна вважати конструктором за замовчуванням для структури *DiskGroup*

2.2. Функція *LoadGroup*, що відповідає за десеріалізацію групи дисків із файлової системи хостової машини.

2.3. Метод *Dump*, що відповідає за серіалізацію групи дисків та їх даних на хостову файлову систему. Цей метод повинен викликатися щоразу після виконання модифікацій групи дисків.

2.4. Метод *StoreBytes*, що записує задані байти на диски, одразу порахувавши відповідні корегуючі коди. Цей метод використовується для запису файла на групу дисків.

2.5. Метод *ReadFile*, що читає файл із дисків за заданим іменем, але при цьому не виконує корекції помилок.

2.6. Метод *Corrupt*, який виконує штучне створення помилок у заданому диску.

2.7. Метод *Restore*, який відновлює дані на заданому диску із використанням збережених корегуючих сум.

Крім роботи із групою дисків, пакет ще передбачає роботу із окремими дисками, що включає структуру *Disk* та функції *NewDisk*, *LoadDisk* і метод *Dump*, що використовуються, відповідно, для ініціалізації нового диска, десеріалізації даних диска із файла на хостовій файловій системі та серіалізації даних на диску у файл.

Також для оперування із сутностями файлів, мінімальну функціональність забезпечує структура *Offsets*, що також надає функцію *NewOffsets*, метод

DumpOffsets та функцію *LoadOffsets*, для ініціалізація, серіалізації та десеріалізації відповідно.

3. Пакет *config* надає дані для конфігурації програми і містить змінну PathPrefix, що дозволяє міняти назву директорії, куди зберігаються дані дискових груп.
4. Пакет *actions* надає функції для обробки окремих дій, які ініціює користувач. Обробник кожної дії знаходиться у окремому файлі та представляє із себе умовний контроллер. Крім цього, кожна дія супроводжується виводом помилок, що виникли при роботі, або повідомленням про успішне її виконання. Визначеними діями є:
    4.1. *clean* — відповідає за очистку даних про дискову групу із хостової файлової системи. Аргументом приймає лише назву дискової групи і реалізується функцією *Clean()*.
    4.2. *corrupt* — відповідає за спотворення даних на диску для штучного моделювання виникнення помилок при зберіганні даних. Аргументами приймає назву дискової групи та номер диска, дані якого потрібно спотворити і реалізується функцією *Corrupt()*.
    4.3. *create* — відповідає за створення нової групи дисків та всіх ресурсів для неї. Аргументами приймає номер групи та кількість дисків у ній і реалізується функцією *Create()*.
    4.4. *info* — надає можливість користувачу переглянути інформацію про групу, зокрема кількість дисків та файлів у ній. Аргументами приймає лише назву дискової групи і реалізується функцією Info().
    4.5. *read* — дозволяє зчитати файл із дискової групи без виконання корекції помилок. Аргументами приймає назву дискової групи та назву файла для зчитування і реалізується функцією *Read()*.
    4.6. *restore* — дозволяє відновити дані із заданого диска із використанням розробленого метода відновлення даних. Аргументами приймає назву дискової групи та номер диска і реалізується функцією *Restore()*.

4.7. *store* — дозволяє зберегти дані ти виконати їх резервування із розробленим методом резервування даних. Аргументами є дискова група та назва файла на хостовій файловій системі і реалізується функцією *Store()*.

5. Пакет *correction*, що реалізує розроблені методи резервування та відновлення даних при їх зберіганні на відновлених носіях.

Така структура програми дозволяє легко її розширяти без внесення багатьох змін у вже існуючі компоненти, що відповідає принципам відкритості-закритості (англ. open-closed principle) та єдиного обов’язку (англ. single responsibility principle) і є хорошою практикою при проектуванні програмних систем.

Крім цього, інтерфейс та внутрішня структура програми є також чітко розділеними, при цьому за інтерфейс відповідають пакети *main* та *actions*, а за внутрішню структуру програми — пакети *diskgroup* та *correction*. Це також дозволяє міняти інтерфейс незалежно від логіки роботи програми та навпаки, наприклад при покращенні алгоритму резервування чи відновлення, чи реалізації додаткових допоміжних дій для користувача, наприклад для відображення більшої інформації про систему. Крім цього, це означає, що інтерфейс за потреби можна міняти, використавши, наприклад, графічний інтерфейс у вигляді додатку чи веб-сторінки. Також внутрішню структуру програми, тобто власне реалізовані методи резервування та відновлення, можна незалежно від інтерфейсу використовувати у ролі зовнішньої бібліотеки, наприклад для вже існуючих програмних засобів для резервування та відновлення даних, для їх покращення.

Як уже згадувалось, через використання саме *Go*, програма може бути легко скомпільована у бінарний файл на будь-якій системі, і тоді розповсюджена на інші системи, на які *Go* навіть не встановлений. Також особливістю структури програми є використання паралельних операцій запису

на файл та зчитування з файлу через природню та невимушену підтримку паралелізму засобами *Go*.

3.3. Розробка інтерфейсу користувача

Для користувачів розроблена програма надає текстовий інтерфейс через командний рядок. Рішення використовувати саме текстовий інтерфейс пов’язане з тим, що при низькорівневих операціях на віддалених комп’ютерних системах графічний інтерфейс є не надто практичним, через низку причин, до яких входять:

1. Ресурсоємність використання графічного інтерфейсу на обчислювальному вузлі.
2. Значне використання ресурсів мережі при віддаленому користуванні. При такому сценарії графічний інтерфейс є швидше зайвою вагою, ніж зручністю.

Крім цього, текстовий інтерфейс є більш зручним для зв’язування програм у конвеєр із різних команд командного рядка та є інколи більш швидким та зручним способом сприйняття інформації.

Після компіляції користувач працює із бінарним файлом *bup*, якому він може передавати різні параметри, і таким чином взаємодіяти із програмою. На рис. 3.1 зображено приклад виконання програми без аргументів.

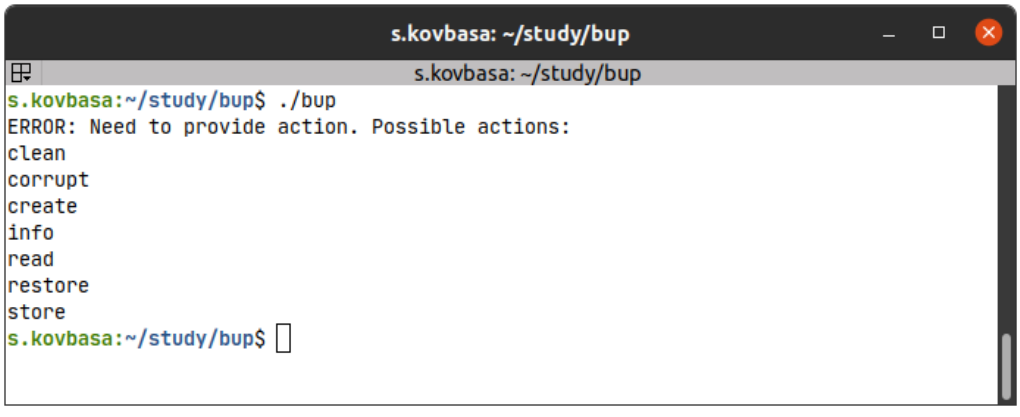

Рис. 3.1. Приклад виконання програми без аргументів

Як бачимо, програма одразу повідомляє про необхідність вибору однієї із реалізованих дій для роботи із програмою. Підтримуваними діями є:

- create — створення нової дискової групи. При цьому програма створює нову папку, що відповідає групі дисків та файли в ній, що відповідають самим дискам, це необхідно для моделювання роботи із віддаленими системами зберігання. Цей етап є необхідним для ініціалізації роботи з програмою і всі інші команди будуть повертати помилку, якщо bup create не була виконана перед ними. Крім цього, можна створити одразу декілька груп та працювати з ними незалежно одна від одної.
- info — відображення інформації про вже створену дискову групу. Інформацією, що відображається, є назва групи, а також кількість дисків та файлів у ній.
- clean — очищення попередньо створеної дискової групи. При цьому група дисків видаляється, що відображається видаленням відповідної допоміжної директорії та файлів дисків. Варто зауважити, що дана операція є невідворотною та тому повинна використовуватись із обережністю.
- store — збереження файлу із хостової системи у попередньо створеній моделі дискової групи. Відповідно, назва файла, що передається для виконання операції має відповідати існуючому файлу. На дисковій групі цей файл зберігається у відповідності до розробленого методу резервування даних. Довільна кількість файлів може бути збережена на створеній дисковій групі та їхній формат не має значення. У дисковій групі файл зберігається під тим же іменем, під яким називався оригінальний файл, тому одним із обмежень є неможливість збереження файлів із однаковим ім’ям, що, проте, ніяк не впливає на можливість перевірки роботи алгоритму резервування та відновлення даних.
- read — зчитування файла із попередньо створеної дискової групи. Спроби прочитати неіснуючий файл закінчаться виводом відповідної помилки. Крім цього, операція передбачає передачу в якості аргумента файл для

виводу на хостовій системі. Саме у цей файл будуть записані поточні дані із моделі дискової групи. Варто зауважити, що цей файл буде створено, якщо він ще не існує, проте якщо такий файл вже є, то розроблена програма тихо його перезапише, і всі попередні дані буде втрачено.

- corrupt — штучне пошкодження одного із дисків дискової групи. Ця операція дозволяє імітувати втрату даних на носіях, для цього вона спотворює дані на вибраному диску із заданої групи. При цьому файли, частини яких там збережено буде пошкоджено, та інформація в них буде спотворена чи взагалі недоступна. Пошкодження диска несе випадковий характер.
- restore — відновлення даних одного із дисків дискової групи. При цьому використовується розроблений метод відновлення даних із використанням попередньо зарезервованих даних при зберіганні файлів. Саме виконання комбінації операцій corrupt та restore дозволяє перевірити правильність та ефективність виконання відновлення даних розробленим методом.

Якщо користувач спробує виконати будь-яку дію, що не зазначена у вищезгаданому переліку, програма виведе відповідну помилку та вкаже, знову ж таки, доступні для користувача дії. На рис. 3.2 показаний приклад саме такого порядку подій, коли користувач намагається виконати неіснуючу операцію ‘asdf’.

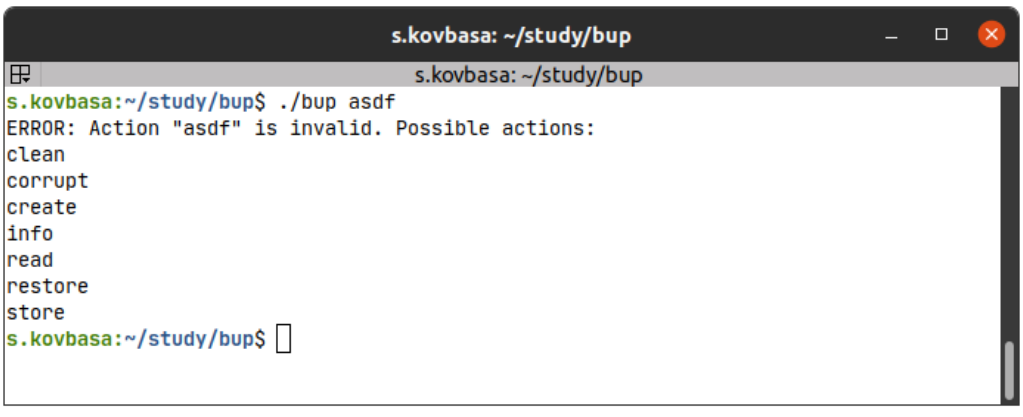

Рис. 3.2. Приклад спроби виконання неіснуючої операції

Розглянемо типовий сценарій користування програмою. Для цього першою дією, яку потрібно виконати, є створення нової групи дисків. На рис. 3.3 показано приклад виконання відповідної команди із як неуспішними спробами та відповідними підказками, так і успішною спробою.

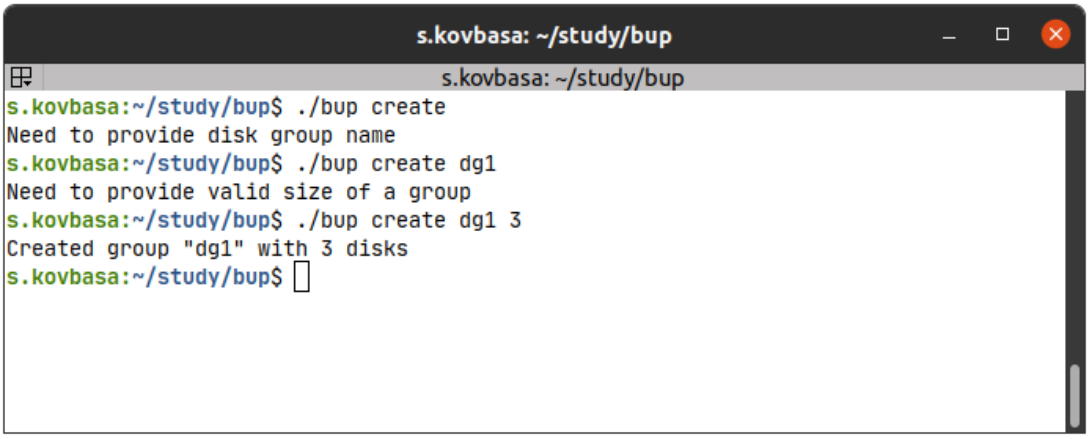

Рис. 3.3. Приклад виконання операції create

Як бачимо, програма виводить допомогу при неправильному чи неповному використанні операції create, а також повідомлення про успішне створення відповідної групи дисків.

Наступним етапом є перевірити створення групи дисків, переглянувши інформацію про неї. На рис. 3.4 зображено приклад виконання команди bup info для цих цілей.

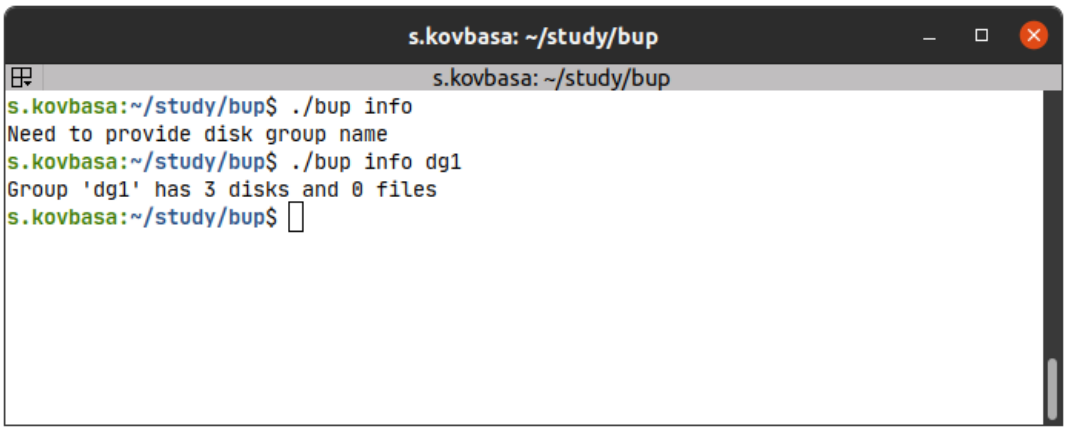

Рис. 3.4. Приклад виконання операції info

Таким чином, як і у попередньому прикладі, програма виводить відповідно помилки та інформацію про успішне її виконання. Саме у цьому випадку видно, що раніше створена група існує та містить 3 диска і (поки що) 0 файлів.

Далі логічним є зберегти декілька файлів на дисковій групі. На рис. 3.5 показано приклад зберігання двох файлів у створеній дисковій групі dg1.

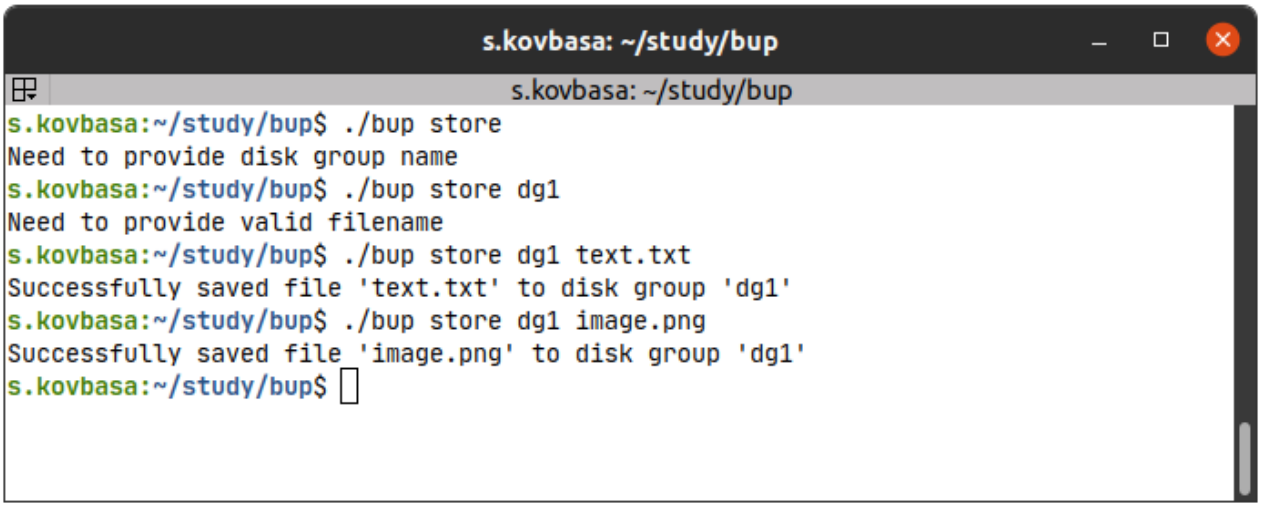

Рис. 3.5. Приклад виконання команди store

Знову ж таки, програма дає зворотній зв’язок при роботі з нею, що відображається виразними повідомленнями про помилки та успішне виконання операції. Варто зауважити, що ці файли (зображення image.png та текстовий файл text.txt) були попередньо збережені на хостовій файловій системі. На рис. 3.6 показано інформацію про ці файли, а також перші декілька рядків текстового файлу, що зберігається. Це дасть більше інформації про правильність виконання операції штучного пошкодження файлів у майбутньому, що, у свою чергу, є необхідним для перевірки ефективності алгоритму відновлення втраченої інформації.

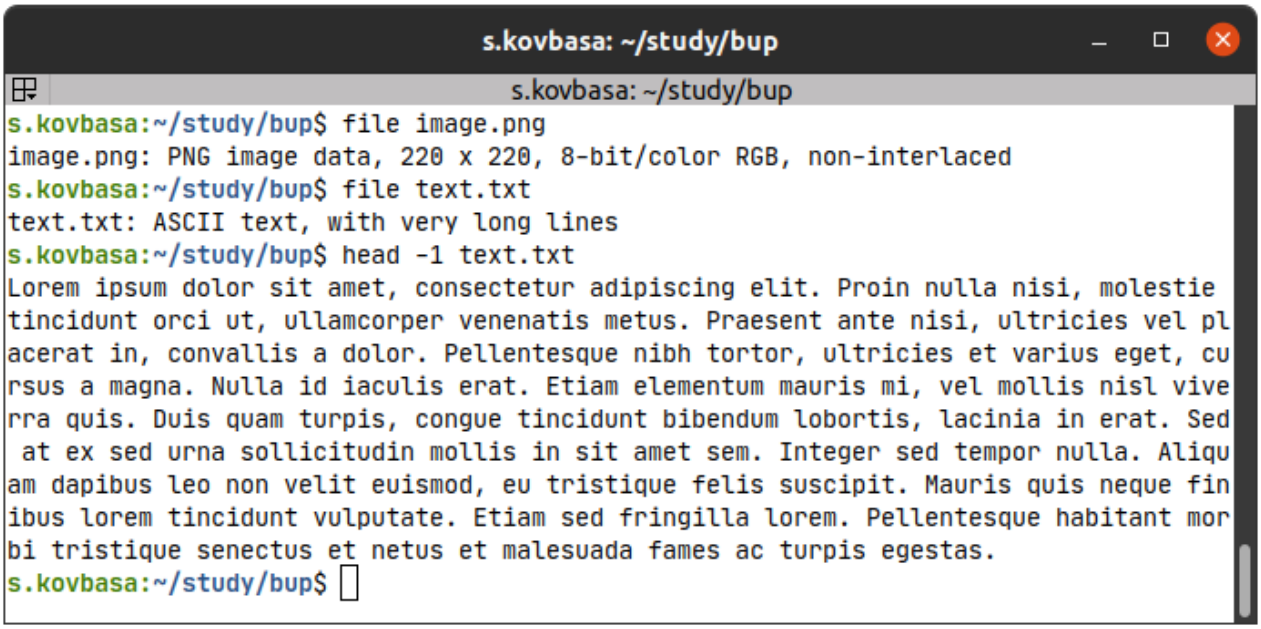

Рис. 3.6. Частина вмісту збережених файлів та інформація про них

Наступним кроком у використанні програми є перевірка правильності збереження файлів. На рис 3.7 для цього показано приклад виконання читання файлів із дискової групи.

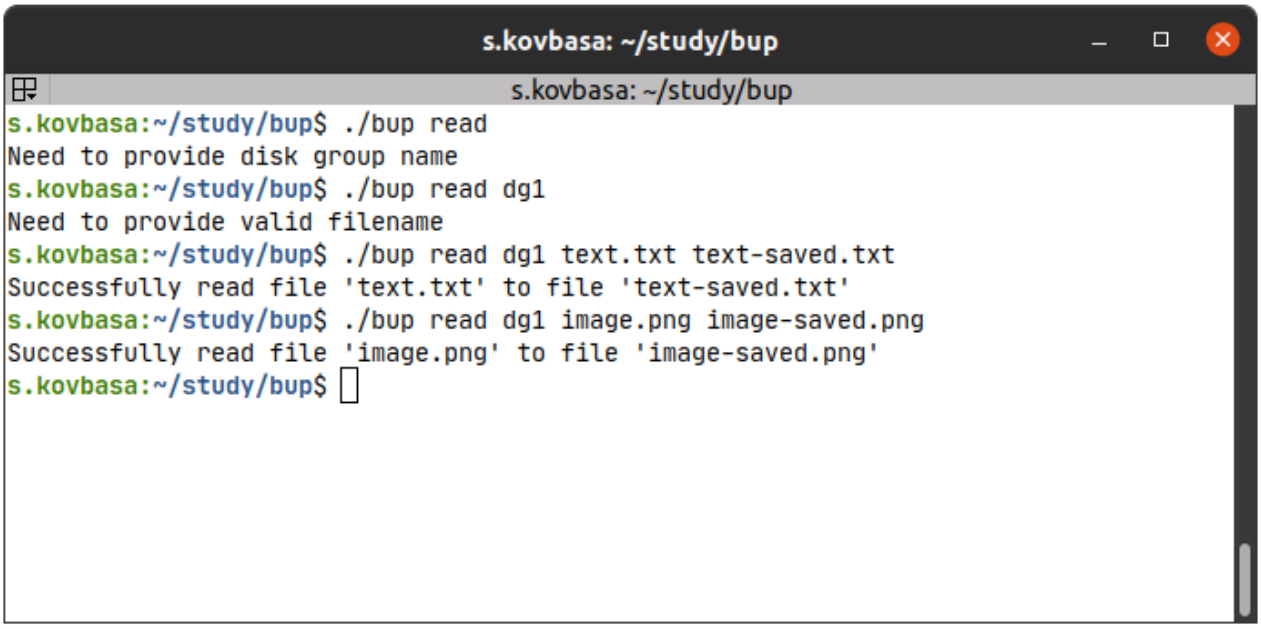

Рис. 3.7. Приклад виконання команди read

Як бачимо, програма і тут надає чіткі повідомлення про помилки, неправильність та успішність її виконання. На рис. 3.8 показано інформацію про ці файли, а також їх порівняння із оригіналами для того, аби впевнитися, що файли повністю однакові.

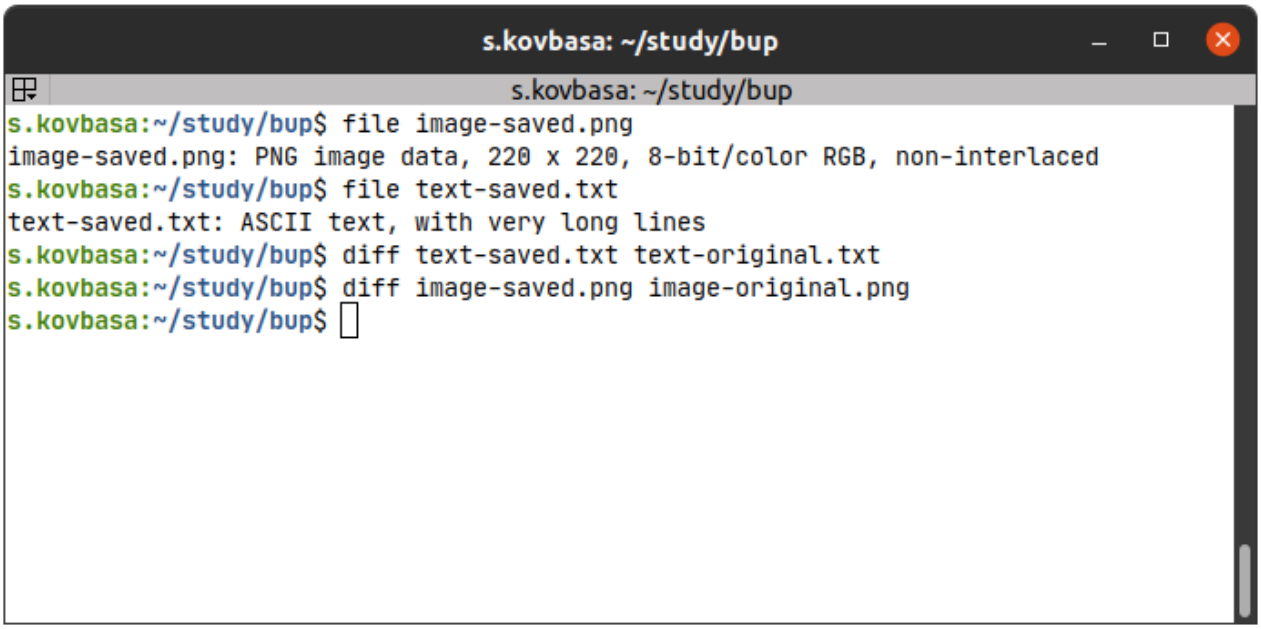

Рис. 3.8. Інформація про збережені на дисковій групі файли

Як бачимо, зчитані файли повністю відповідають оригіналам, що свідчить про правильність виконання операцій зберігання та читання. На рис. 3.9 крім цього, показано зміну відображення інформації про дискову групу, яка тепер показує, що група dg1 містить два збережених файла.

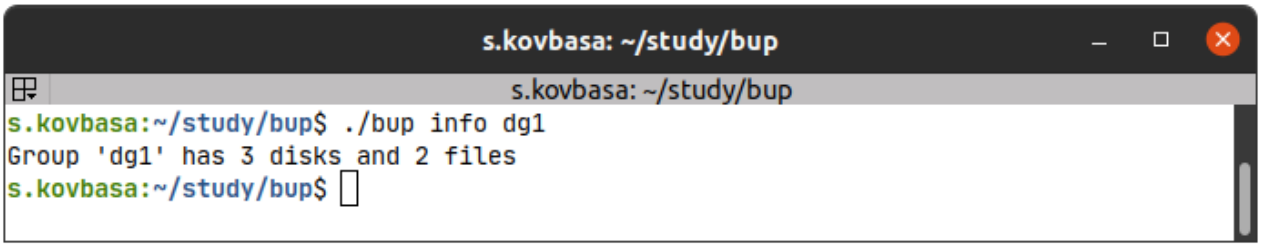

Рис. 3.9. Приклад виконання команди info після збереження двох файлів у дисковій групі

Всі вищезгадані операції дозволяють перевірити правильність моделювання групи дисків, та операцій зчитування з них та запису на них. Головною метою цієї програми є саме перевірка розроблених методів резервування та відновлення даних на практиці. Для цього треба пройти ще через два етапи — спотворення та відновлення даних.

Далі необхідно імітувати виникнення пошкодження даних на дисках. На рис. 3.10 показано приклад виконання цієї операції для диска під номером 1 (нумерація дисків починається з 0, також варто нагадати, що у вже створеній групі є 3 диска).

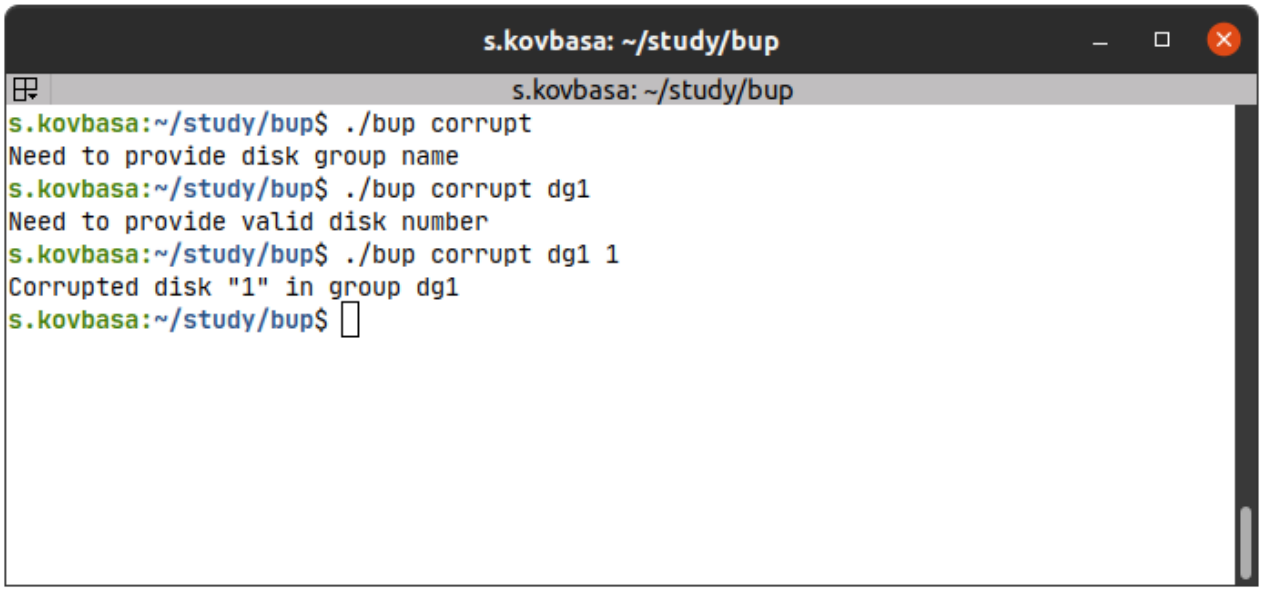

Рис. 3.10. Приклад виконання команди corrupt

При неправильному використанні команди corrupt, програма виводить зрозумілі повідомлення про помилки, а при правильному використанні — повідомлення про її успішне завершення, що є корисним для користувача.

На рис. 3.11 показано повторення операцій зчитування, перевірки вмісту раніше збережених файлів та їх порівняння із оригіналами, для того, аби запевнитись в їх пошкодженні.

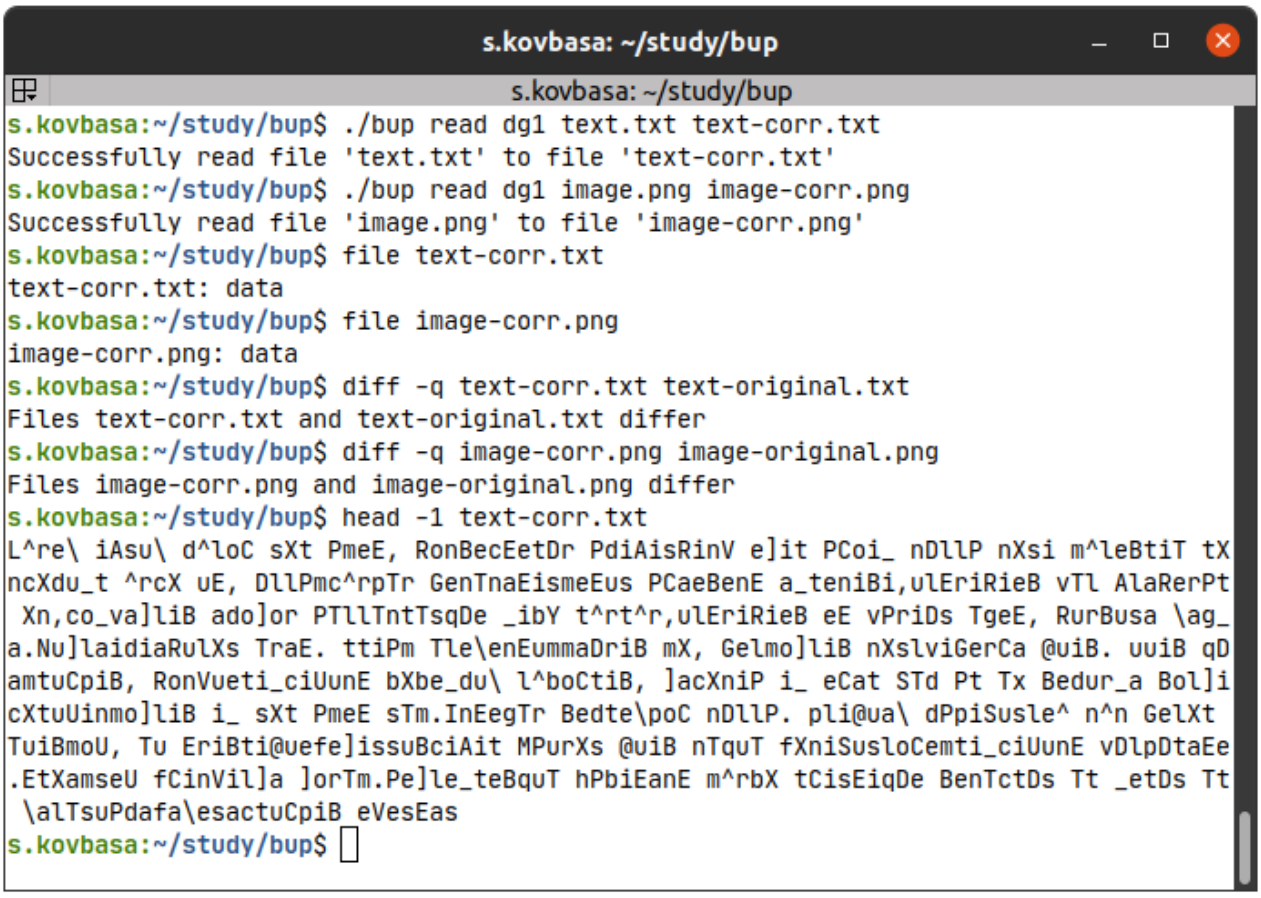

Рис. 3.11. Вміст збережених файлів після пошкодження диска 1

Як бачимо, після повторного читання збережених файлів після пошкодження дисків, збережені файли також є пошкодженими. Це проявляється у декількох виглядах. По-перше, утиліта file тепер не розпізнає файли як текстовий та зображення відповідно. Це означає, що деякі байти було пошкоджено, так що метаінформацію файлів було пошкоджено теж. По-друге, повторно зчитані файли відрізняються від оригінальних. По-третє, наглядно пошкодження диска можна побачити при друку першого рядка текстового файлу, де замість правильного тексту, виводиться пошкоджений зі спотвореними символами. Це все може свідчити про правильне виконання команди corrupt.

Наступним кроком є безпосереднє відновлення даних із втраченого диска. На рис. 3.12 показаний приклад виконання відновлення диска за допомогою розробленого алгоритма, вбудованого у написану програму.

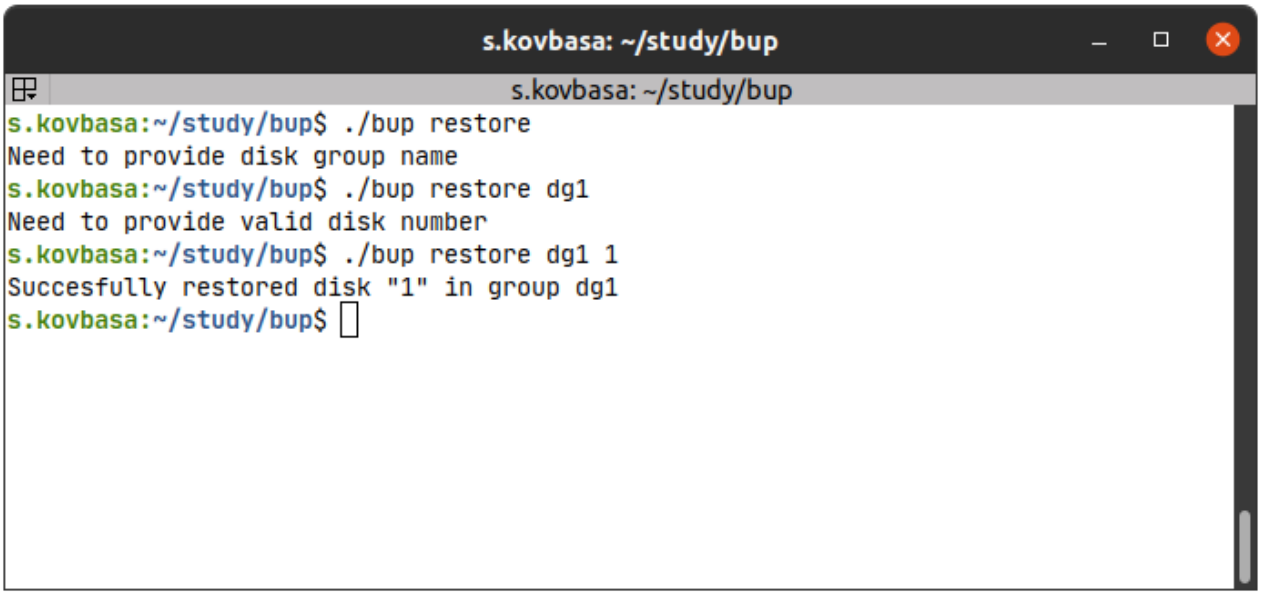

Рис. 3.12. Приклад виконання команди restore

Знову ж таки, очевидним є той факт, що програма і при виконанні цієї команди є виразною як при неправильному, так і при правильному користуванні. На рис. 3.13 показано ще одне повторення команд зчитування раніше збережених файлів, перевірки їх вмісту та порівняння з оригіналами для того, аби переконатись в успішному відновленні раніше пошкоджених даних.

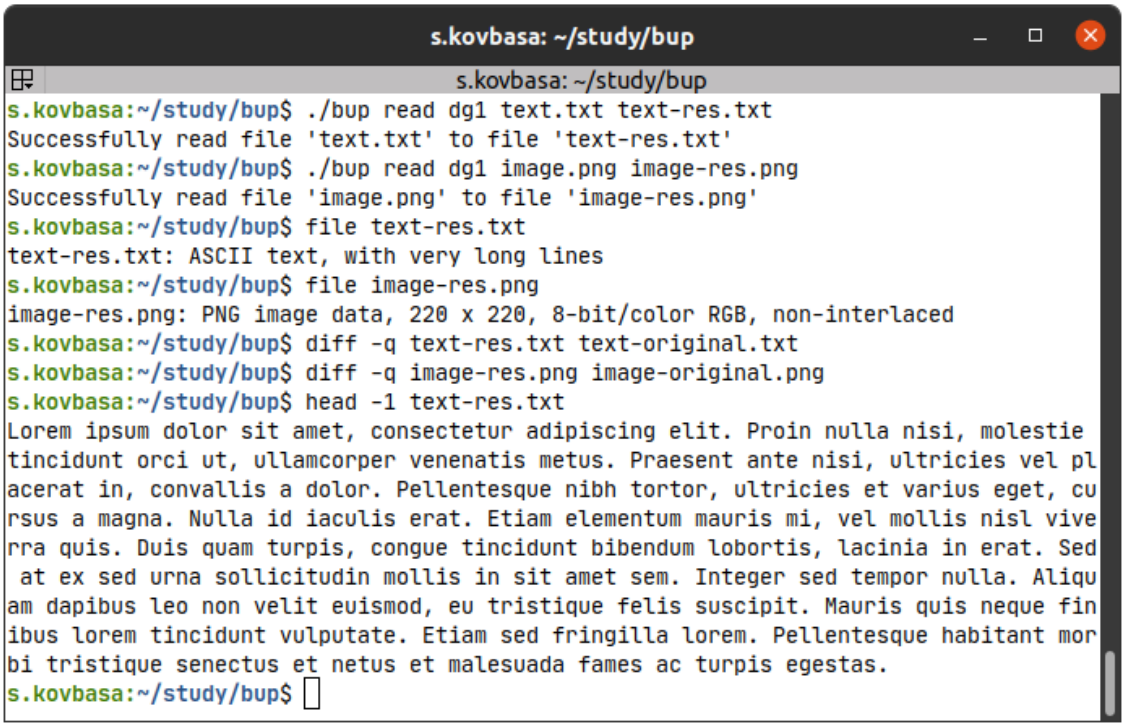

Рис. 3.13. Вміст збережених файлів після відновлення диска 1

Як бачимо, після відновлення диска 1, раніше збережені файли було повністю відновлено також. Це можна бачити із того, що типи файлів, ідентифіковані утилітою file, знову повернулися у нормальний вигляд - тепер це знову текстовий файл та зображення. Крім цього, знову зчитані файли нічим не відрізняються від оригіналів і наочно відновлення можна бачити на першому рядку текстового файла, який знову виглядає як непошкоджений, чистий текст.

Останнім опціональним етапом у теоретичному сценарії використання розробленої програми є очистка дискових груп, що передбачає видалення з файлової системи моделей дискової групи та дисків. На рис. 3.14 показано приклад виконання команди очистки.

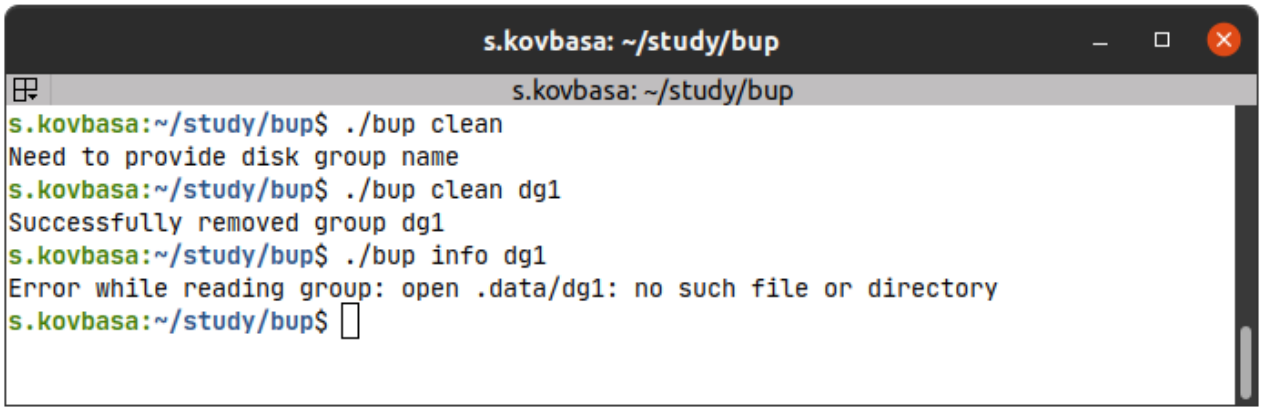

Рис. 3.14. Приклад виконання команди clean

Ця команда не є виключенням, і при її виконанні програма виводить явні помилки при її використанні або повідомлення про успішне її виконання. Видно й те, що перевірка інформації про видалену групу дисків повертає помилку, що говорить, що така група дисків не існує.

Висновки до розділу 3

У результаті розробки програмної реалізації методу резервування та відновлення даних із трьох втрачених носіїв, можна зробити наступні висновки:

1. Аналіз мов програмування показав, що для цієї задачі найкраще підходить мова програмування Go, що дозволяє на низькому рівні керувати даними програми, і при цьому забезпечує високий рівень надійності та швидкості виконання всіх операцій.

2. Реалізовано програму, що моделює використання віддалених носіїв даних та реалізує запропонований метод резервування та відновлення даних із трьох втрачених носіїв на мові Go. Реалізована програма дозволяє виконати весь сценарій користування віддаленими носіями, включаючи створення нової групи дисків, а також запису та читання файлів з них. Також реалізована програма дозволяє виконати штучне пошкодження обраних дисків та їх відновлення розробленими методами.

3. Детально описано структуру розробленого програмного продукту, вказано модулі та функції, зони їх відповідальності та взаємозв’язки між ними. Показано, що структура розробленого програмного продукту відповідає кращим практикам побудови програмного забезпечення.

4. Детально показано приклад виконання відновлення даних за розробленим методом, що дає змогу переконатися у правильності розробленого алгоритму та у тому, що він несе практичну цінність.

РОЗДІЛ 4
РОЗРОБКА СТАРТАП-ПРОЕКТУ

4.1. Опис ідеї проекту

Сьогодні вже існує безліч рішень, що виконують резервування та відновлення даних при їх зберіганні на віддалених носіях. Деякі із них були згадані у розділі 1, крім того, існують безліч апаратних та програмних реалізацій RAID та схожих відкритих алгоритмів, що також є комерційними. Варто зауважити, що розроблений програмний продукт не можна розглядати як повноцінний продукт через те, що він створений для демонстрації запропонованого методу резервування та відновлення даних, що зберігаються на віддалених носіях. Незважаючи на це, ідею за створеним програмним продуктом можна розвивати, та із достатньою кількістю планування, його можна розвинути у ефективний, конкурентоспроможний продукт.

В рамках цієї роботи, за її темою можна можна реалізувати декілька ідей. Розроблений метод має потенціал для успішного втілення у наступних продуктах:

1. Апаратний контролер дисків для виконання їх резервування та відновлення.
2. Програмний продукт для виконання резервування та відновлення даних на основі вже існуючої інфраструктури віддалених носіїв.

У цьому розділі буде описано розробку стартап-проекту, що втілює програмний продукт для виконання резервування та відновлення даних, що зберігаються на віддалених носіях. У таблиці 4.1 викладено зміст ідеї, можливі напрямки її застосування, основні вигоди, які може отримати користувач з ідеї та відмінності ідеї від аналогів та замінників.

Таблиця 4.1

Опис ідеї стартап-проекту

Зміст ідеї	Напрямки застосування	Вигоди для користувача
Програмний продукт для виконання резервування та відновлення даних	Виконання автоматичного резервування даних при запису на віддалені носії та автоматичного відновлення при їх читанні.	Можливість одноразово налаштувати політику виконання резервування та відновлення даних та значно зменшити ризики, пов’язані із можливою втратою даних на віддалених носіях, а також зменшити витрати часу на контроль процесів резервування та відновлення.
	Виконання резервування та відновлення даних з віддалених носіїв на вимогу користувача.	Можливість самостійно керувати виконанням резервування та відновлення даних для можливості більш точного та продуктивного використання резервування та відновлення даних та значного зменшення ризиків, пов’язаних із можливою втратою даних на віддалених носіях.
	Виконання автоматичної синхронізації файлів із системи користувача на віддалені носії.	Економія часу та зусиль для завантаження файлів на віддалені носії.

Аналіз потенційних техніко-економічних переваг ідеї (чим відрізняється від існуючих аналогів та замінників). Проектами-конкурентами є програмні засоби для резервування даних Backblaze, Zoolz та Dropbox

Таблиця 4.2

Визначення сильних, слабких та нейтральних характеристик ідеї проекту

№	Техніко-економічні характеристики ідеї	Мій проект	Конкуренти			W	N	S
			1	2	3			
1	Мультиплатформенність	+	+	+	+		+	
2	Можливість зберігати будь-які файли користувача	+	+	+	+		+	
2	Відновлення даних при одночасній відмові трьох носіїв	+	-	-	-	+		
3	Синхронізація файлів із файлової системи користувача	+	-	-	+		+	
4	Виконання резервування та відновлення даних на вимогу користувача	+	+	+	-		+	
5	Можливість роботи у терміналі	+	-	-	+	+		
6	Наявність власних віддалених носіїв	-	+	+	+			+

Визначений перелік слабких, сильних та нейтральних характеристик та властивостей ідеї потенційного товару є підґрунтям для формування його конкурентоспроможності. Як видно з таблиці 4.2, основними перевагами розроблюваного продукту є можливість відновлення даних при одночасній відмові трьох носіїв, можливість синхронізації файлів із файлової системи та можливість роботи у терміналі, на що і варто зробити ставку. В майбутньому варто врахувати основні недоліки розроблюваного продукту, а саме розробити власну інфраструктуру віддалених носіїв для більш зручного використання більшістю користувачів.

4.2. Технологічний аудит ідеї проекту

Таблиця 4.3

Технологічна здійсненність ідеї проекту

№	Ідея проекту	Технології її реалізації	Наявність технології	Доступність технологій
1	Програмний продукт для виконання резервування та відновлення даних	Мова програмування Go	Є в наявності	Доступні безкоштовно
2		Алгоритми відновлення та резервування даних	Є в наявності	Доступні безкоштовно
3		Мова програмування JavaScript та фреймворк React	Є в наявності	Доступні безкоштовно

Обрана технологія реалізації ідеї проекту: розробка основної програми на мові програмування Go, та графічного веб-інтерфейсу на мові програмування JavaScript та фреймворку React, алгоритми резервування та відновлення даних, в тому числі запропонований в поточній роботі.

4.3. Аналіз ринкових можливостей запуску стартап-проекту

Визначення ринкових можливостей, які можна використати під час ринкового впровадження проекту, та ринкових загроз, які можуть перешкодити реалізації проекту, дозволяє спланувати напрями розвитку проекту із урахуванням стану ринкового середовища, потреб потенційних клієнтів та пропозицій проектів-конкурентів. В таблиці 4.4 наведена попередня характеристика потенційного ринку для розроблюваного стартап-проекту.

Таблиця 4.4

Попередня характеристика потенційного ринку стартап-проекту

№	Показники стану ринку	Характеристика
1	Кількість головних гравців, од	більше 20
2	Динаміка ринку (якісна оцінка)	Зростає
3	Наявність обмежень для входу (вказати характер обмежень)	Велика кількість конкурентів,
4	Специфічні вимоги до стандартизації та сертифікації	Необхідність отримання SSL сертифікату, стандартизацією віддалених сховищ займається організація Cloud Standards Coordination.

За попереднім оцінюванням можна зробити висновок, що ринок є привабливим, проте доволі складним для входу.

Надалі визначаються потенційні групи клієнтів, їх характеристики, та формується орієнтовний перелік вимог до товару для кожної групи. Усі ці дані вказані у таблиці 4.5.

Таблиця 4.5

Характеристика потенційних клієнтів стартап-проекту

№	Потреба, що формує ринок	Цільова аудиторія	Відмінності у поведінці різних потенційних цільових груп клієнтів	Вимоги споживачів до товару
1	Стрімке збільшення кількості даних, що використовуються	Корпоративні клієнти, датацентри, дослідники та науковці в галузі великих даних, компанії що займаються аналітикою	Кількість використовуваних даних, вимоги до надійності їх зберігання	Потреба у зручному інтерфейсі, високій швидкості доступу до даних
2	Необхідність забезпечувати максимальні рівні надійності для довготривалого зберігання даних	Приватні компанії, компанії із великою кількістю архівних даних, компанії-власники чутливих даних	Вимоги до надійності зберігання даних, частота доступу до них	Потреба у надвисокій надійності зберігання даних

Після визначення потенційних груп клієнтів у таблицях 4.6 та 4.7 проводиться аналіз ринкового середовища.

Таблиця 4.6

Фактори загроз

№	Фактор	Зміст загрози	Можлива реакція компанії
1	Висока конкуренція	На даний момент існує велика кількість конкурентів, які в основному повторюють функціонал одне одного, тому є загроза не отримати достатньої уваги від клієнтів	Швидка реакція команди розробників на побажання потенційних клієнтів, постійне оновлення та розширення функціоналу
2	Вузький спектр функціоналу і можливостей	На ринок треба вийти якомога швидше, тому до цього моменту повинні бути реалізовані всі основні функції для забезпечення можливості конкуренції та розширенню клієнтської бази	Затримка моменту виходу на ринок, з метою розширення базового набору функціоналу на момент старту поширення програми
3	Економічний	В силу обмеженості в фінансовому плані, можливі накладки, так як до моменту виходу на ринок компанія не дає жодного прибутку	Побудова чіткого бізнес-плану із залученням інвесторів

Таблиця 4.7

Фактори можливостей

№	Фактор	Зміст можливості	Можлива реакція компанії
1	Залучення інвестицій	Склавши чіткий бізнес-план, можна залучити інвесторів, що дозволить розширити штат і прискорити момент виходу на ринок	Збільшити штат, прискорити вихід на ринок, вийти на період самоокупності раніше
2	Розширення клієнтської бази	З допомогою продуманої PR-кампанії, можна переманити клієнтів конкурентів запропонувавши більш вигідні умови	Спланувати рекламну кампанію, проводити презентації та семінари на тему резервування та відновлення даних, в яких для демонстрації буде використано розроблюваний продукт
3	Заключення контрактів на роботи	Укладання договорів на надання послуг резервування та відновлення даних на віддалених носіях	Отримати зв’язки з потенційними клієнтами та використовувати розроблювану програму для виконання резервування та відновлення даних на замовлення

У таблиці 4.8 проводиться аналіз пропозиції: визначаються загальні риси конкуренції на ринку.

Таблиця 4.8

Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	В чому проявляється дана характеристика	Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)
Складність входу на ринок	Ринок не є новим та велика кількість конкурентів уже укріпили свої позиції, завели зв'язки та мають налагоджену клієнтську базу	Пропонувати більш вигідні умови (навіть з можливістю працювати без прибутку), аналізувати та впроваджувати нові розробки в сфері резервування даних раніше конкурентів, якісна рекламна кампанія
Загальнодоступність інформації	Алгоритми резервування та відновлення даних є переважно оприлюдненими та доступними для всіх.	Критично важливо готувати оновлення в мінімальний після оприлюднення алгоритмів термін, щоб мати перевагу перед конкурентами
Схожість продуктів-замінників	Так як продукти на обраному ринку виконують фактично одну і ту ж функцію, важко суттєво вирізнятися серед конкурентів	Експериментувати з ідеями для нових оновлень, впроваджувати нові рішення в мінімальний термін

У таблиці 4.9 проводиться більш детальний аналіз умов конкуренції в галузі.

Таблиця 4.9

Аналіз конкуренції в галузі за М. Портером

Складові аналізу	Прямі конкуренти в галузі	Потенційні конкуренти в галузі	Постачальники	Клієнти	Товари-замінники
	Backblaze, Dropbox, Zoolz та інші	fex.net, Google Drive, iCloud	Amazon, Microsoft Azure	Приватні та відкриті компанії	Open Source-рішення
Висновки:	Існує велика кількість різноманітних рішень у конкурентів	Є маса додатків, що можуть виконувати такі ж операції	Постачальниками є світові лідери у сфері хмарних систем	Більшість клієнтів уже давно на ринку та мають налагоджені контакти з постачальниками	Є безкоштовні рішення розроблені ентузіастами для виконання операцій резервування даних

Отже, принципова можливість виходу на ринок з огляду на конкурентну ситуацію існує, проте є складною.

Таблиця 4.10

Обґрунтування факторів конкурентоспроможності

№	Фактор конкурентоспроможності	Обґрунтування (наведення чинників, що роблять фактор для порівняння конкурентних проектів значущим)
1	Зручний інтерфейс	Так як проект на етапі розробки, можна врахувати переваги та недоліки конкурентів, щоб до виходу на ринок мати зручніший ніж в інших інтерфейс користувача
2	Забезпечення різних інтерфейсів	Більшість з конкурентів розробляють веб-додаток або додатки із графічним інтерфейсом, тоді як для багатьох задач значно зручнішим є саме текстовий інтерфейс. Це надасть можливість розширити клієнтську базу
3	Швидке реагування на відгуки	На етапі розробки неможливо передбачити всі нюанси застосування. Завдяки забезпеченню швидкого відгуку, можна використовувати користувачів у якості бета-тестувальників, що дозволяє максимально швидко адаптувати додаток під потреби реальних клієнтів
4	Забезпечення високої надійності збереження даних	Для багатьох людей критичною є саме надійність зберігання даних, тоді як більшість систем віддаленого зберігання надають лише базовий рівень надійності.

Таблиця 4.11

Порівняльний аналіз сильних та слабких сторін

№	Фактор конкурентоспроможності	Бали 1-20	Рейтинг товарів-конкурентів у порівнянні з розроблюваним продуктом						
			-3	-2	-1	0	1	2	3
1	Зручний інтерфейс	15			✓				
2	Забезпечення різних інтерфейсів	19	✓						
3	Швидке реагування на відгуки	10				✓			
4	Забезпечення високої надійності збереження даних	16			✓				

Таблиця 4.12

SWOT-аналіз стартап-проекту

Сильні сторони: Зручний інтерфейс Забезпечення різних інтерфейсів Можливість врахування досвіду конкурентів Швидке реагування на побажання клієнтів Забезпечення високої надійності збереження даних	Слабкі сторони: Велика кількість конкурентів Наявність open source рішень Недовіра великих клієнтів
Можливості: Отримати контракти на резервування та відновлення віддалених носіїв	Загрози: Програти конкуренцію open source проектам Не відбити частину клієнтів від уже наявних конкурентів

Таблиця 4.13

Альтернативи ринкового впровадження стартап-проекту

№	Альтернатива ринкової поведінки	Ймовірність отримання ресурсів	Строки реалізації
1	Відкриття проекту та розповсюдження в рамках open source ліцензії	Низька, так як в основному це будуть пожертви ентузіастів. Можливість продажу компанії у випадку зацікавленості з боку великих компаній	Практично миттєво
2	Розповсюдження проекту через конференції, публічні виступи	Висока, оскільки публічними заходами можна поширити ідею проекту та завести знайомства з потенційними інвесторами та клієнтами	До року, з можливістю продовження
3	Участь в тендерах і укладення договорів з державними установами	Середня, так як на це впливають економічні, політичні, особисті мотиви і не можна гарантувати абсолютну прозорість та добросовісну конкуренцію на цьому ринку	Кілька місяців, в залежності від наявності запиту з боку держави

4.4. Розробка ринкової стратегії проекту

Таблиця 4.14

Вибір цільових груп потенційних споживачів

№	Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнтовний попит в межах цільової групи (сегменту)	Інтенсивність конкуренції в сегменті	Простота входу у сегмент
1	Датацентри	Готові сприйняти негайно	Продукт необхідний не всім	Висока конкуренція	Складний вхід
2	Приватні компанії	Не всі користувачі готові прийняти	Продукт необхідний не всім	Конкуренція середня	Вхід помірно-складний
3	Навчальні та дослідницькі програми	Не всі користувачі готові прийняти продукт	Продукт необхідний не всім	Конкуренція низька	Вхід не є складний
Які цільові групи обрано: Потрібно працювати зі всіма цільовими групами, перевагу надати приватним компаніям					

Таблиця 4.15

Визначення базової стратегії розвитку

№	Обрана альтернатива розвитку проекту	Стратегія охоплення ринку	Ключові конкурентос-проможні позиції відповідно до обраної альтернативи	Базова стратегія розвитку
1	Розвиток шляхом маркетингової компанії та публічної демонстрації можливостей продукту	Впливати на обрані цільові групи в порядку значимості та перспектив	Виступ на конференціях, вільне розповсюдження для навчально-дослі дницьких цілей	Оперативно оновлювати продукт відповідно до побажань клієнтів

Таблиця 4.16

Визначення базової стратегії конкурентної поведінки

№	Чи є проект «першопрохідц ем» на ринку?	Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?	Чи буде компанія копіювати основні характеристики товару конкурента, і які?	Стратегія конкурентної поведінки
1	Ні, проект не є «першопрохідц ем»	Необхідно рухатись у різних напрямках, як шукати нових споживачів, так і відбирати існуючих у конкурентів	Через схожість функціоналу, компанія буде копіювати основні характеристики у конкурентів	Пропонувати схожий функціонал на кращих умовах, оперативно оновлювати функціонал відповідно до нових алгоритмів

Таблиця 4.17

Визначення стратегії позиціонування

№	Вимоги до товару цільової аудиторії	Базова стратегія розвитку	Ключові конкуренто-спроможні позиції власного стартап-проекту	Вибір асоціацій, які мають сформувати комплексну позицію власного проекту (три ключових)
1	Висока частота оновлення, широкий функціонал, кросплатформеність	Розширити штат, досліджувати новини щодо виходу нових алгоритмів	Оперативне виправлення недоліків та впровадження побажань	Надійне сховище, зручність для адміністраторів, швидке відновлення втрачених даних

4.5. Розробка маркетингової програми стартап-проекту

Таблиця 4.18

Визначення ключових переваг концепції потенційного товару

№	Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити)
1	Резервування та відновлення даних на віддалених носіях	Надійний та швидкий метод резервування та відновлення даних	Можливість відновлювати дані одночасно із трьох носіїв, у разі втрати доступу до них.
2	Постійне резервування всіх локальних файлів	Можливість синхронізувати локальні файли із віддаленим сховищем	Можливість доступу через командний рядок без необхідності встановлення графічного інтерфейса

Таблиця 4.19

Опис трьох рівнів моделі товару

<table>
<tr><td>Рівні товару</td><td colspan="2">Сутність та складові</td></tr>
<tr><td>I. Товар за задумом</td><td colspan="2">Програмний продукт для виконання резервування та відновлення даних на віддалених носіях, із можливістю синхронізації локальних файлів та зручним інтерфейсом через командний рядок</td></tr>
<tr><td rowspan="6">II. Товар у реальному виконанні</td><td>Властивості/характеристики</td><td>Розмір</td></tr>
<tr><td>1. Модуль виконання процедур резервування та відновлення даних</td><td>20 МБ</td></tr>
<tr><td>2. Back-end сервер для підтримки веб-додатку</td><td>50 МБ</td></tr>
<tr><td>3. Front-end скрипт для формування інтерфейсу користувача</td><td>10 МБ</td></tr>
<tr><td colspan="2">Якість: внутрішнє тестування програми, логування збоїв та система зворотнього зв’язку для повідомлення про неналежну роботу програми</td></tr>
<tr><td colspan="2">Пакування: продаж електронних ключів-ліцензій, онлайн-підписок</td></tr>
<tr><td rowspan="2">III. Товар із підкріпленням</td><td colspan="2">Марка: назва організації-розробника та назва товару</td></tr>
<tr><td colspan="2">До продажу: програмний код</td></tr>
<tr><td colspan="2">Після продажу: електронна версія виконуваного файлу програми з ключем</td></tr>
<tr><td colspan="3">За рахунок чого потенційний товар буде захищено від копіювання:
прив’язка копії програмного продукту до конкретного ПК та активація програми шляхом введення ліцензійного ключа, або шляхом надання послуг без передачі програмного продукту замовнику</td></tr>
</table>

Таблиця 4.20

Визначення меж встановлення ціни

№ п/п	Рівень цін на товари-замін ники	Рівень цін на товари-анало ги	Рівень доходів цільової групи споживачів	Верхня та нижня межі становлення ціни на товар/послугу
1	10$ - 100$	10-100	>10000$/місяць	50-100

Таблиця 4.21

Формування системи збуту

№ п/п	Специфіка закупівельної поведінки цільових кліє- нтів	Функції збуту, які має виконувати постачальник товару	Глибина каналу збуту	Оптимальна система збуту
1	Покупка ліцензії на продукт або договір про надання послуг без передачі ПО до замовника	Розміщення ПО в центрах електронної дистрибуції	Канал збуту однорівневи й, через роздрібну дистрибуцію	Вертикальна (право власності залишається у розробника)

Таблиця 4.22

Концепція маркетингових комунікацій

№ п/п	Специфіка поведінки цільових клієнтів	Канали комунікацій, якими користуються цільові клієнти	Ключові позиції, обрані для позиціонування	Завдання рекламно го повідомл ення	Концепція рекламного звернення
1	Купують товар на вимогу	Тематичні зустрічі, конференції, презентації тощо	Надання послуг резервування даних розробленим продуктом, продаж ПЗ	Демонстр ація основних функцій розробле ного продукту	Охоплення аудиторії, пояснення функцій та можливостей ПО та його переваг

Висновки до розділу 4

У результаті розробки стартап-проекту для розробленого програмного продукту, можна зробити наступні висновки:

1. Наведено загальний опис ідеї проекту для виходу на ринок, описаний функціонал проекту, визначено основних конкурентів та співставлено функціонал конкурентів для визначення переваг розроблюваного продукту над обраними конкурентами.

2. Проведено технічний аудит проекту і визначено можливості реалізації програмного продукту.

3. Проаналізовано ринкові можливості для запуску проекту, порівняно перспективи запуску з конкурентами, встановлено план та ринкову стратегію розвитку продукту, визначено цільові групи та способи просування проекту.

4. Розроблено маркетингову програму для просування продукту на ринку, описано способи та основні канали збуту, визначено пріоритетні цільові групи та маркетингові повідомлення для розширення клієнтської бази.

В підсумку, було отримано стартап-проект для запуску розробленого програмного продукту на ринок, отримано навички створення стартап-проектів, побудови маркетингової стратегії та аналізу обраного ринку.

ВИСНОВКИ

Дана магістерська дисертація присвячена теоретичному обґрунтуванню та розробці нових методів резервування та відновлення даних, що зберігаються на віддалених носіях, включаючи метод відновлення двох втрачених віддалених носіїв із використанням лінійних кодів, його модифікації із використанням діагональних сум та покращення останнього методу, що дозволяє відновлювати до трьох втрачених носіїв за спрощеною процедурою відновлення.

Основними науковими та практичними результатами роботи є наступні:

1. Проведено огляд існуючих рішень побудови систем віддалених носіїв, проаналізовано їх рівні забезпечення надійності таких систем. Оглянуто критерії ефективності алгоритмів резервування та відновлення даних, та у їхньому контексті виконано огляд існуючих рішень для резервування та відновлення даних. Показано недоліки та переваги цих методів, проаналізовано та теоретично обґрунтовано методи корекції помилок, що використовуються при відновленні даних з носіїв. Проаналізовано особливості кодів корекції помилок при їх використанні для відновлення втрачених носіїв. Показано, що сучасні методи резервування та відновлення помилок ефективно вирішують проблему відновлення помилок із одного втраченого носія, проте алгоритми, що вирішують проблему відновлення помилок із декількох втрачених носіїв є неефективними з точки зору часової складності виконання, що спонукає до розробки нових методів резервування та відновлення помилок із кількох втрачених носіїв.

2. Розроблено та досліджено метод резервування та відновлення даних із двох втрачених носіїв з використанням лінійних сум. Показано переваги такого підходу, що полягають у спрощенні процедури відновлення, та його недоліки, що полягають у надмірній надлишковості цього методу. Як наслідок запропоновано модифікацію розробленого методу, що використовує діагональні суми і дозволяє значно зменшити його надлишковість при використанні тих же простих операцій.

3. На основі запропонованого методу розроблено спрощений метод для резервування та відновлення даних із трьох віддалених носіїв із використанням діагональних сум. Досліджено властивості цього методу, показано, що він забезпечує високу швидкість роботи за рахунок зменшення часової складності при незначному збільшенні надлишковості даних на $2(n+1)$. Показано, що розроблений метод використовує лише прості операції додавання за модулем 2, що робить його простим у виконанні як програмно, так і апаратно. Виконано порівняння розробленого методу резервування та відновлення трьох втрачених носіїв з використанням діагональних сум із кодами Ріда-Соломона, показано, що запропонований метод має більшу ефективність виконання процедури відновлення даних.

4. Розроблено програмний продукт, який дозволяє експериментально перевірити правильність розробленого методу відновлення трьох втрачених носіїв. Розроблена програма надає засоби для моделювання роботи із віддаленими носіями, виникнення помилок та відновлення втрачених дисків із використанням розроблених методів.

5. Із допомогою розробленого програмного продукта виконано практичне дослідження запропонованого методу резервування та відновлення даних. Використовуючи розроблений програмний продукт, експериментально показано правильність та ефективність виконання резервування та відновлення даних із втрачених носіїв із використанням запропонованого методу.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Коваленко, В. Эволюция и проблемы Grid / В. Коваленко, Д. Корягин // Открытые системы. — 2003. — №1. — С. 27—33.

2. Ron Levine. NAS Advantages: A VARs View [Електронний ресурс] / Ron Levine // Infostor. — 1998. — Режим доступу до ресурсу: https://www.infostor.com/index/articles/display/55961/articles/infostor/volume-2/issue-4/news-analysis-trends/nas-advantages-a-vars-view.html.

3. Magoutis, K. Structure and Performance of the Direct Access File System / K. Magoutis, S. Addetia, A. Fedorova, M. Seltzer, J. Chase, A. Gallatin, R. Kisley, R. Wickremesinghe, E. Gabber // Proceedings of the General Track: 2002 USENIX Annual Technical Conference. — 2002. — С. 1—14.

4. Alexandros G. Dimakis. Network Coding for Distributed Storage Systems / Alexandros G. Dimakis, P. Brighten Godfrey, Yunnan Wu, Martin Wainwright and Kannan Ramchandran // IEEE Transactions on Information Theory. — Vol 56. — 2010.

5. Lakshman, A. Cassandra: A decentralized structured storage system. / A. Lakshman, P. Malik // SIGOPS Oper. Syst. Rev. — 2010. — № 44 — С. 35—40.

6. Ma, G. Performance Evaluation of Storage Systems Based on Network-Attached Disks / G. Ma, A. Khaleel, A. Narasimha // IEEE Transactions on Parallel and Distributed Systems. — Vol 11, № 9. — 2000. — С. 956 — 968.

7. Тормасов,А. Г. Модель распределенного хранения данных с регулируемой избыточностью / А. Г. Тормасов, М. А. Хасин, Ю. И. Пахомов // Электронный журнал «Исследовано в России». — 2001. — С.14—22.

8. Vrable M. Filesystem backup to the cloud / Vrable M., Savage S., Voelker G. // ACM Trans. on Storage (ToS). — 2009. — Vol. 54. — № 12. — С.11—17.

9. M. Blaum. Partial MDS Codes and Their Application to RAID Type of Architectures / M. Blaum, James Lee Hafner, Steven Hetzler // IEEE Trans. Inf. Theory. — 2013. — Vol.59. — №7. — С. 4510 — 4519.

10. Ivanov D. Paladin: Secure and redundant cloud storage / Heuert U., Ivanov D. // Herald of the Merseburg University of Applied Sciences. — Merseburg: Elbe Drucketei Wittenberg GmbH. — 2011. — № 8. — С.220—228.

11. Corbett P. Row-diagonal parity for double disk failure. / English D., Goel A., Grcanac T., Kleiman S., Leong D., Sankar S. // In Proceedings of the Third USENIX Conference on File and Storage Technologies, USA. — 2004. — С.1—14.

12. Patterson D. Case of Redundant Array of Inexpensive Disk (RAID) / Patterson D., Gibson G., Katz R. /// Berkeley: University of California. — 1987. — С. 238.

13. Understanding RAID [Електронний ресурс] // cru-inc.com — 2014 — Режим доступу до ресурсу: https://www.cru-inc.com/data-protection-topics/understanding-raid/

14. Selecting the best RAID level [Електронний ресурс] // Sun StorageTek SAS RAID HBA Installation Guide. — 2010. — Режим доступу до ресурсу: https://docs.oracle.com/cd/E19691-01/820-1847-20/appendixf.html

15. Fan, B. Diskreduce: Raid for data-intensive scalable computing. / Fan, B., Tantisiriroj, W., Xiao, L., Gibson, G. // PDSW ’09: Proceedings of the 4th AnnualWorkshop on Petascale Data Storage. — 2009. — ACM, Portland, USA. — С. 6—10.

16. Marcus, Evan. Blueprints for high availability (2, illustrated ed.) / Marcus, E., Stern, H. // John Wiley and Sons. — 2003. — С. 167.

17. Natarajan, Ramesh. RAID 2, RAID 3, RAID 4 and RAID 6 Explained with Diagrams. / Natarajan, Ramesh // TheGeekStuff.com. — 2011.

18. RAID 5 Data Recovery FAQ [Електронний ресурс] // VantageTech.com. Vantage Technologies. — 2014. — Режим доступу до ресурсу: https://vantagetech.com/faq/raid-5-recovery-faq.html.

19. Sun StorageTek SAS RAID HBA Installation Guide, Appendix F: Selecting the Best RAID Level: RAID 6 Arrays. [Електронний ресурс] // Docs.Oracle.com. — 2010. — Режим доступу до ресурсу:

https://docs.oracle.com/cd/E19691-01/820-1847-20/appendixf.html#50515995_51002.

20. X.Kai. A Construction of New MDS Symbol-Pair Codes / X.Kai, S. Zhu, P.Li // IEEE Trans. Inf. Theory — 2015 — Vol.61 — №11 — С. 5828—5834.

21. Charles Wang. Forward Error-Correction Coding / Charles Wang, Dean Sklar, Diana Johnson // Crosslink. The Aerospace Corporation — 2002.

22. Katina Kralevska. Applied Erasure Coding in Networks and Distributed Storage / Katina Kralevska // ResearchGate: Thesis for the degree of Philosophiae Doctor. — 2018. — С. 7.

23. Berger C.R. Optimizing Joint Erasure and Error-Correction Coding for Wireless Packet Transmissions / Willett P., Pattirati K. // IEEE Transactions on wireless communications. — Vol.7. — №11. — 2008. — С. 4586—4595.

24. Коляда К.В. Метод відновлення даних при їх розподіленому зберіганні на віддалених сховищах / К.В. Коляда, В.О. Романкевич, М.М. Орлова, О.П. Марковський // Комп’ютерно-інтегровані технології: освіта, наука, виробництво. — № 40. — 2020. — С.44—50.

25. Коляда К.В. Метод резервування та відновлення втрачених даних в глобальних мережах / К.В. Коляда, О.П. Марковський, В.Г. Саверченко, А.І. Торошанко // Телекомунікаційні та інформаційні технології. — № 1 (66). — 2020. — С.4—14.

26. N. Bardis, O. Markovskyi. Method for Cloud Storage Data Recovery with Limited Loss of Access / N. Bardis, O. Markovskyi // Proc. of 4-th International Conference on Mathematics and Computers in Sciences and Industry — 2017. — С.55—61.

27. Иванов Д. Г. Организация резервирования в системах распределенного хранения данных / Иванов Д. Г. // Вісник Національного технічного університету України ”КПІ” Інформатика, управління та обчислювальна техніка. — 2012. — № 56. — С.160—164.

28. Cipra, Barry Arthur. The Ubiquitous Reed–Solomon Codes / Cipra, Barry Arthur // SIAM News, 26 (1) — 1993.

29. Ирвин Дж., Харль Д. Передача данных в сетях: инженерный подход // БХВ-Петербург — 2002. — С.448.

30. Klove T., Korzhik V. Error Detecting Codes: General Theory and Their Application in Feedback Communication Systems. / Klove T., Korzhik V. // Norwell, MA: Kluwer — 1995. — С.433.

31. Блейхут Р. Теория и практика кодов, контролирующих ошибки // Мир — 1986. — С.576.

32. Морелос-Сарагоса Р. Искусство помехоустойчивого кодирования. Методы, алгоритмы, применение // Техносфера. — 2006. — С. 320.

33. K.V. Rashmi, B.Nihar, K. Ramchandran. Information-Theoretically Secure Erasure Codes for Distributed Storage. / K.V. Rashmi, B.Nihar, K. Ramchandran // IEEE Trans. Inf. Theory — Vol.64. — №3 — 2018 — С.1621—1645.

34. Дансмор Б., Скандьер Т. Справочник по телекоммуникационным технологиям // Вильямс. — 2004. — С.640.

35. Zhang, Z. Does erasure coding have a role to play in my data center? / Zhang, Z., Deshpande, A., Ma, X., Thereska, E., Narayanan, D. // Tech. Rep. MSR-TR-2010-52, Microsoft Research. — 2010.

36. Марковский А.П., Мулки Ахмед Яссин Ал Бадайнех, Корниец Е.В. Обнаружение многократных ошибок передачи данных с использованием контрольной суммы // Труды 8-й международной научно-технической конференции ’’Современные информационные и электронные технологии — Одеса, 2007. — С.193.

37. Vishwanath, K.V. Characterizing cloud computing hardware reliability. / Vishwanath, K.V., Nagappan, N. // Proceedings of the 1st ACM Symposium on Cloud Computing. Indianapolis, USA — 2010. — С.193—204.

38. Иванов Д.Г. Метод восстановления данных на основе скошенных матриц в системах распределенного хранения / Луцкий Г.М., Иванов Д.Г. // Известия высших учебных заведений. Проблемы полиграфии и издательского

дела. Информационные технологии. УПИПК МГУП им. И.Федорова. — 2013. — № 2. — С. 47—52.

39. Wing Q. End-to-End Error-Correcting Codes on Networks with Worst-Case Bit Errors / Q. Wing, S. Jaggi // IEEE Transaction on Information Theory. — 2018. — Vol. 64. — No. 6. — С. 4467—4479.

40. Plank, J. Optimizing cauchy reed-solomon codes for fault-tolerant network storage applications. / Plank, J., Xu, L. // Network Computing and Applications. — 2006. — С.173—180.

41. Mortuza A. A Parametric Approach to List Decoding of Reed-Solomon Codes Using Interpolation / Mortuza A., Kuijper M. // IEEE Transactions on information theory. —2011. — Vol.57. — № 10.

ДОДАТОК А
ТЕКСТ ПРОГРАМИ

```
package main

import (
   "flag"
   "fmt"
   "os"
   "sort"
   "strings"

   "s.kovbasa/bup/actions"
)

var actionsMap = map[string]func() error{
   "create":  actions.Create,
   "store":   actions.Store,
   "corrupt": actions.Corrupt,
   "read":    actions.Read,
   "restore": actions.Restore,
   "clean":   actions.Clean,
   "info":    actions.Info,
}

func main() {
   flag.Parse()

   availActions := make([]string, 0, len(actionsMap))
   for a := range actionsMap {
       availActions = append(availActions, a)
   }
   sort.Strings(availActions)
   availActionsStr := strings.Join(availActions, "\n")

   action := flag.Arg(0)
   if action == "" {
       fmt.Fprintf(
           os.Stderr,
           "ERROR: Need to provide action. Possible actions:\n%v\n",
           availActionsStr,
       )
       os.Exit(1)
   }

   if doAction, hasAction := actionsMap[action]; hasAction {
       if err := doAction(); err != nil {
```

```
            fmt.Fprint(os.Stderr, err)
            os.Exit(1)
        }
        os.Exit(0)
    }

    fmt.Fprintf(
        os.Stderr,
        "ERROR: Action \"%s\" is invalid. Possible actions:\n%v\n",
        action,
        availActionsStr,
    )
    os.Exit(1)
}

package diskgroup

import (
    "fmt"
    "io/ioutil"
)

type Disk struct {
    Data []byte
}

func NewDisk() *Disk {
    data := make([]byte, 0)
    return &Disk{data}
}

func LoadDisk(filename string) (*Disk, error) {
    data, err := ioutil.ReadFile(filename)
    if err != nil {
        return nil, err
    }
    return &Disk{data}, nil
}

func (d *Disk) Dump(filename string) error {
    if err := ioutil.WriteFile(filename, d.Data, 0664); err != nil {
        return err
    }
    return nil
}

func (d *Disk) WriteByte(b byte) error {
    d.Data = append(d.Data, b)
```

```
   return nil
}

func (d *Disk) ReadByteFrom(offset uint64) (byte, error) {
   if offset >= uint64(len(d.Data)) {
       return 0, fmt.Errorf("Trying to read unavailable data, offset >
len(data)\n")
   }
   return d.Data[offset], nil
}

package diskgroup

import (
   "fmt"
   "io/ioutil"
   "os"
   "path"
   "regexp"

   "s.kovbasa/bup/config"
   "s.kovbasa/bup/correction"
)

type DiskGroup struct {
   Name        string
   Size        uint64
   Disks       []*Disk
   FileOffsets Offsets
}

func NewGroup(name string, size uint64) *DiskGroup {
   disks := make([]*Disk, 0, size)
   for i := 0; i < int(size); i++ {
       disks = append(disks, NewDisk())
   }
   return &DiskGroup{
       Name:        name,
       Size:        size,
       Disks:       disks,
       FileOffsets: NewOffsets(),
   }
}

func LoadGroup(name string) (*DiskGroup, error) {
   groupPath := path.Join(config.PathPrefix, name)
   files, err := ioutil.ReadDir(groupPath)
   if err != nil {
```

```
        return nil, err
    }
    filenames := make([]string, 0)

    r := regexp.MustCompile("disk[0-9]+")
    for _, f := range files {
        if f.IsDir() {
            return nil, fmt.Errorf("Bad disk group: found dir '%s' insted of disk
file", f.Name())
        }
        if match := r.MatchString(f.Name()); !match {
            continue
        }
        filenames = append(filenames, f.Name())
    }

    size := uint64(len(filenames))
    disks := make([]*Disk, 0, size)
    for _, filename := range filenames {
        disk, err := LoadDisk(path.Join(groupPath, filename))
        if err != nil {
            return nil, err
        }
        disks = append(disks, disk)
    }
    offsets, err := LoadOffsets(groupPath)
    if err != nil {
        return nil, err
    }

    return &DiskGroup{
        Name:        name,
        Size:        size,
        Disks:       disks,
        FileOffsets: offsets,
    }, nil
}

func (dg *DiskGroup) Dump() error {
    groupPath := path.Join(config.PathPrefix, dg.Name)
    if err := os.MkdirAll(groupPath, 0775); err != nil {
        return err
    }
    for i, disk := range dg.Disks {
        filename := path.Join(groupPath, fmt.Sprintf("disk%d", i))
        if err := disk.Dump(filename); err != nil {
            return err
        }
```

```
    }
    if err := dg.FileOffsets.DumpOffsets(groupPath); err != nil {
        return err
    }

    return nil
}

func (dg *DiskGroup) StoreBytes(name string, data []byte) error {
    maxOffset := uint64(0)

    for key, offset := range dg.FileOffsets {
        if key == name {
            return fmt.Errorf("File '%s' is already stored\n", name)
        }
        if offset > maxOffset {
            maxOffset = offset
        }
    }
    for i, b := range data {
        idx := uint64(i) + maxOffset
        disk := dg.Disks[idx%dg.Size]
        _ = disk.WriteByte(b)

        corrByte := correction.GetChecksum(b)
        _ = disk.WriteByte(corrByte)
    }
    dg.FileOffsets[name] = maxOffset + uint64(len(data))
    return nil
}

func (dg *DiskGroup) ReadFile(name string) ([]byte, error) {
    fileOffset, found := dg.FileOffsets[name]
    if !found {
        return nil, fmt.Errorf("File '%s' not stored\n", name)
    }
    prevOffset := uint64(0)
    for _, offset := range dg.FileOffsets {
        if offset < fileOffset && offset > prevOffset {
            prevOffset = offset
        }
    }
    data := make([]byte, 0, fileOffset-prevOffset)
    for i := prevOffset; i < fileOffset; i++ {
        disk := dg.Disks[i%dg.Size]
        byteOffset := i / dg.Size
        b, err := disk.ReadByteFrom(byteOffset * 2)
        if err != nil {
```

```
            return nil, err
        }
        data = append(data, b)
    }
    return data, nil
}

func (dg *DiskGroup) Corrupt(diskNo uint64) error {
    if int(diskNo) >= len(dg.Disks) {
        return fmt.Errorf("No disk with No %d exists is a group %s\n", diskNo,
dg.Name)
    }
    disk := dg.Disks[diskNo]
    mask := correction.GetCorruptionByte()
    for i, b := range disk.Data {
        if i%2 == 1 {
            continue
        }
        disk.Data[i] = mask ^ b
    }
    return nil
}

func (dg *DiskGroup) Restore(diskNo uint64) error {
    if int(diskNo) >= len(dg.Disks) {
        return fmt.Errorf("No disk with No %d exists is a group %s\n", diskNo,
dg.Name)
    }
    disk := dg.Disks[diskNo]
    for i, b := range disk.Data {
        if i%2 == 1 {
            continue
        }
        disk.Data[i] = correction.CorrectByte(b, disk.Data[i+1])
    }
    return nil
}

package diskgroup

import (
    "encoding/gob"
    "os"
    "path"
)

var offsetsFilename = ".offsets"
```

```
type Offsets map[string]uint64

func NewOffsets() Offsets {
   return Offsets(make(map[string]uint64))
}

func (o Offsets) DumpOffsets(dirpath string) error {
   filename := path.Join(dirpath, offsetsFilename)
   f, err := os.Create(filename)
   if err != nil {
       return err
   }
   defer f.Close()

   encoder := gob.NewEncoder(f)
   if err := encoder.Encode(map[string]uint64(o)); err != nil {
       return err
   }
   return nil
}

func LoadOffsets(dirpath string) (Offsets, error) {
   var offsets Offsets
   filename := path.Join(dirpath, offsetsFilename)
   f, err := os.Open(filename)
   if err != nil {
       return nil, err
   }
   defer f.Close()
   decoder := gob.NewDecoder(f)

   if err := decoder.Decode(&offsets); err != nil {
       return nil, err
   }
   return offsets, nil
}

package correction

import (
   "math/rand"
   "time"
)

func GetCorruptionByte() byte {
   m := []byte{1, 1, 1, 0, 0, 0, 0, 0}
   rand.Seed(time.Now().UnixNano())
   rand.Shuffle(len(m), func(i, j int) { m[i], m[j] = m[j], m[i] })
```

```
   var maskByte byte
   for i, bit := range m {
       maskByte += bit * (1 << i)
   }
   return maskByte
}

func GetChecksum(b byte) byte {
   return ^b
}

func CorrectByte(b, c byte) byte {
   return ^c
}

package config

var PathPrefix = ".data/"

package actions

import (
   "flag"
   "fmt"
   "os"
   "path"

   "s.kovbasa/bup/config"
)

func Clean() error {
   name := flag.Arg(1)
   if name == "" {
       return fmt.Errorf("Need to provide disk group name\n")
   }
   if err := os.RemoveAll(path.Join(config.PathPrefix, name)); err != nil {
       return fmt.Errorf("Error while removing group: %v\n", err)
   }

   fmt.Printf("Successfully removed group %s\n", name)
   return nil
}

package actions

import (
   "flag"
```

```
   "fmt"
   "strconv"

   "s.kovbasa/bup/diskgroup"
)

func Corrupt() error {
   name := flag.Arg(1)
   if name == "" {
       return fmt.Errorf("Need to provide disk group name\n")
   }

   diskNo, err := strconv.ParseUint(flag.Arg(2), 10, 64)
   if err != nil {
       return fmt.Errorf("Need to provide valid disk number\n")
   }

   dg, err := diskgroup.LoadGroup(name)
   if err != nil {
       return fmt.Errorf("Error while reading group: %v\n", err)
   }

   if err := dg.Corrupt(diskNo); err != nil {
       return fmt.Errorf("Error while corrupting disk: %v\n", err)
   }

   if err := dg.Dump(); err != nil {
       return fmt.Errorf("Error while dumping group: %v\n", err)
   }

   fmt.Printf("Corrupted disk \"%d\" in group %s\n", diskNo, name)
   return nil
}

package actions

import (
   "flag"
   "fmt"
   "strconv"

   "s.kovbasa/bup/diskgroup"
)

func Create() error {
   name := flag.Arg(1)
   if name == "" {
       return fmt.Errorf("Need to provide disk group name\n")
```

```
    }

    size, err := strconv.ParseUint(flag.Arg(2), 10, 64)
    if err != nil {
        return fmt.Errorf("Need to provide valid size of a group\n")
    }

    dg := diskgroup.NewGroup(name, size)
    if err := dg.Dump(); err != nil {
        return fmt.Errorf("Error while dumping group: %v\n", err)
    }

    fmt.Printf("Created group \"%s\" with %d disks\n", name, size)
    return nil
}

package actions

import (
    "flag"
    "fmt"

    "s.kovbasa/bup/diskgroup"
)

func Info() error {
    name := flag.Arg(1)
    if name == "" {
        return fmt.Errorf("Need to provide disk group name\n")
    }

    dg, err := diskgroup.LoadGroup(name)
    if err != nil {
        return fmt.Errorf("Error while reading group: %v\n", err)
    }

    fmt.Printf(
        "Group '%s' has %d disks and %d files\n",
        dg.Name,
        dg.Size,
        len(dg.FileOffsets),
    )
    return nil
}

package actions

import (
```

```
   "flag"
   "fmt"
   "io/ioutil"

   "s.kovbasa/bup/diskgroup"
)

func Read() error {
   name := flag.Arg(1)
   if name == "" {
       return fmt.Errorf("Need to provide disk group name\n")
   }

   filename := flag.Arg(2)
   if filename == "" {
       return fmt.Errorf("Need to provide valid filename\n")
   }

   filenameOut := flag.Arg(3)
   if filenameOut == "" {
       return fmt.Errorf("Need to provide valid output filename\n")
   }

   dg, err := diskgroup.LoadGroup(name)
   if err != nil {
       return fmt.Errorf("Error while reading group: %v\n", err)
   }

   data, err := dg.ReadFile(filename)
   if err != nil {
       return fmt.Errorf("Unable to read file: %v\n", err)
   }

   if err := ioutil.WriteFile(filenameOut, data, 0664); err != nil {
       return fmt.Errorf("Unable to write to output file: %v\n", err)
   }

   fmt.Printf("Successfully read file '%s' to file '%s'\n", filename,
filenameOut)
   return nil
}

package actions

import (
   "flag"
   "fmt"
   "strconv"
```

```
   "s.kovbasa/bup/diskgroup"
)

func Restore() error {
   name := flag.Arg(1)
   if name == "" {
       return fmt.Errorf("Need to provide disk group name\n")
   }

   diskNo, err := strconv.ParseUint(flag.Arg(2), 10, 64)
   if err != nil {
       return fmt.Errorf("Need to provide valid disk number\n")
   }

   dg, err := diskgroup.LoadGroup(name)
   if err != nil {
       return fmt.Errorf("Error while reading group: %v\n", err)
   }

   if err := dg.Restore(diskNo); err != nil {
       return fmt.Errorf("Error while restoring disk: %v\n", err)
   }

   if err := dg.Dump(); err != nil {
       return fmt.Errorf("Error while dumping group: %v\n", err)
   }

   fmt.Printf("Succesfully restored disk \"%d\" in group %s\n", diskNo, name)
   return nil
}

package actions

import (
   "flag"
   "fmt"
   "io/ioutil"

   "s.kovbasa/bup/diskgroup"
)

func Store() error {
   name := flag.Arg(1)
   if name == "" {
       return fmt.Errorf("Need to provide disk group name\n")
   }
```

```
    filename := flag.Arg(2)
    if filename == "" {
        return fmt.Errorf("Need to provide valid filename\n")
    }

    data, err := ioutil.ReadFile(filename)
    if err != nil {
        return fmt.Errorf("Error while reading file: %v\n", err)
    }

    dg, err := diskgroup.LoadGroup(name)
    if err != nil {
        return fmt.Errorf("Error while reading group: %v\n", err)
    }

    if err := dg.StoreBytes(filename, data); err != nil {
        return fmt.Errorf("Unable to store file: %v\n", err)
    }

    if err := dg.Dump(); err != nil {
        return fmt.Errorf("Error while dumping group: %v\n", err)
    }

    fmt.Printf("Successfully saved file '%s' to disk group '%s'\n", filename,
name)
    return nil
}
```